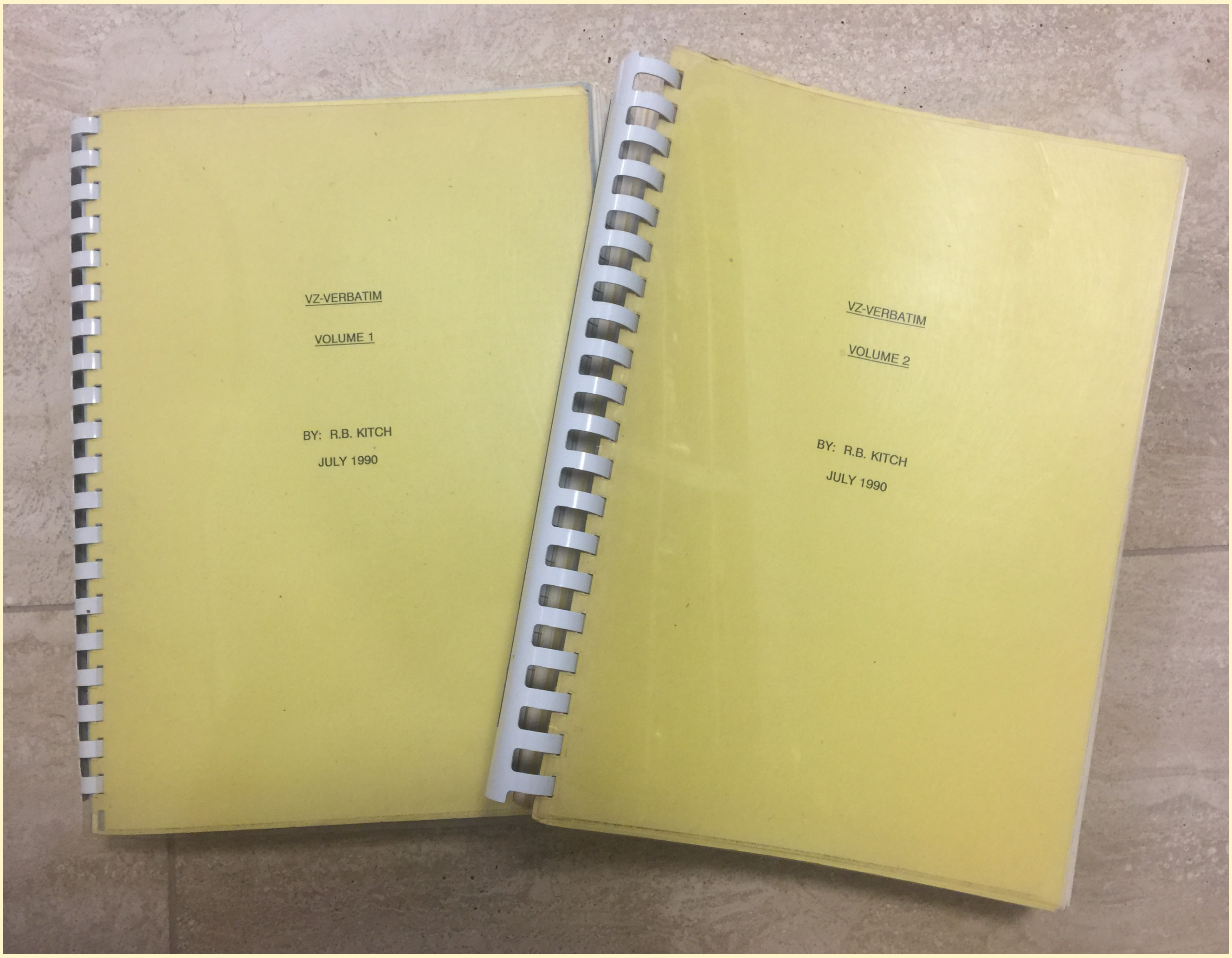
Bob Kitch 1990

VZ—VERBATIM

(A Collection of Magazine and
Technical Articles for
VZ Computers 1981 to 1990)

Volume 2 Peripherals, Reviews, Programming and Technical Bulletins



Compiled by Bob Kitch 1981 to 1990
Scanned March 2021
Brisbane Australia for All VZ Users
E: rbkitch@hotmail.com
M: +61 (0) 400 083 465

COMPILERS GUIDE FOR VZ USERS

Bob Kitch
Brisbane March 2021

***VZ-Verbatim** is a research resource for the DSE VZ00 and VZ300 micro-computers marketed in Australasia during the 1980's - in the pre-PC and post-TRS80/System 80 eras. Many young (and old) computer users cut their digital teeth on these Z80-based machines. A number of VZ User Groups also sprang up, held meetings and produced Newsletters. There was a huge thirst for knowledge, enthusiasm, learning, coding and general learning about "things digital" centred upon the VZ.*

All of the information in this compilation is long out-of-print and quite difficult to obtain. It may not be sold or recompiled into any other format without my express permission. Note the highly practical electronic and computing information that was offered in technical magazines of this era.

An information companion to VZ-Verbatim is the "Bob Kitch's VZ Scrap Book" that contains thirty technical contributions I made to magazines and various User Groups Newsletters during 1985 to 1990. Approximately 25 BASIC and ASSEMBLER ASCII listings are provided in that Directory. These articles were about learning and encouraging VZ Users to develop digital skills and interests.

VZ-Verbatim was a last-Century response to an information demand to encourage a new generation of digital enthusiasts in the pre-WWW era. It was compiled during 1985 to 1990 but with articles going back to 1981. The original format was as loose A4 Master Sheets wherein specific photocopies could be returned by snail mail to interested and puzzled VZ Users. As interest in 8-bit computers waned in early 1990's, a lone copy of VZ-Verbatim (as two volumes) was made (pictured on cover). It is in the last month these volumes have come to hand, been scanned at 400dpi and converted to pdf's.

As a late incarnation of the 8-bit microcomputer era, the Video Technology/DSE VZ200/300 was highly influential in homes throughout Australasia and under other names elsewhere in the World. A fair level of interest remains amongst enthusiasts in Vintage Computer Groups and Emulators Users. A number of now middle-aged men, were young enthusiasts that learned about computing in the 1980's and still use the VZ for largely nostalgic reasons. I note that a remarkable number of these young enthusiasts are now employed in the IT industry. These enthusiasts are instrumental in maintaining Z80 emulators and hardware, have added more convenient I/O peripherals than the contemporary cassette and floppies and have added memory capabilities beyond 64K. Tape and disk software has been converted to more durable digital formats.

Preserving and providing ready access the "Lump" of VZ technical information, software images, operating hardware and emulators is regarded as a priority. This compilation is part of that "VZ Lump".

Bob Kitch
Brisbane, Queensland, Australia
E: rbkitch@hotmail.com
M: +61 (0) 400 083 465

Structure of Volumes

Following on the blue pages is a complete listing of all articles contained within Volumes 1 and 2.

This is shown in the original list format that was frequently updated and circulated to VZ Users.

Pages 12 to 14 of that list is included for completeness. These pages are a list of books on BASIC, Assembler and the Z80. Most of these are available on-line as e-books in pdf format.

The yellow pages detail the various sections within the volumes.

Volume 1 contains software articles categorised as

- **Utilities**
- **Games**
- **Business**

Volume 2 contains

- **Hardware Peripherals**
- **Software Reviews**
- **Software Advertisements**
- **Hardware Reviews**
- **General Programming**
- **DSE Technical Bulletins.**

These volumes were derived from 400dpi scans of second generation photocopies of the original bound articles and were delivered in Adobe Acrobat pdf format.

Using Adobe Acrobat Pro 2017 each page was edited and enhanced involving

- character recognition to provide editable text and images
- text and images de-skewed
- font replaced with document fonts for sharpening

VZ-VERBATIM

VOLUME 2

BY: R.B. KITCH

JULY 1990

LISTING OF VZ200/300 MAGAZINE ARTICLES

AS AT 31 JULY 1990

Since its introduction in early 1983, over three hundred articles on the VZ-200 and 300 have appeared in the magazines. Some articles review the hardware and others describe peripherals. Some excellent games have been published and a very useful set of utility routines has emerged.

This bibliography for the VZ computer is a must for the serious VZ-User.

Compiled by:-

R.B. KITCH, 7 Eurella St., Kenmore, Qld. 4069. Phone: (07)378-3745.
PLEASE ADVISE OF ANY ADDITIONAL ARTICLES ..or.. CHANGES, ALTERATIONS OR BUGS IN LISTINGS to assist other Users.

The numbers in brackets are the number of sheets in each article. A dash (-) indicates that the article is on the same sheet as the item above.

If Users wish to obtain copies of the articles referred to in this bibliography, they may -

i) contact me for copies ..or..
ii) buy back copies of the magazine from the distributor ..or..
iii) borrow from your local library.

I can supply copies FOR YOUR OWN USE ONLY at 20c. per sheet. Kindly add postage to your request as follows:

No. of Sheets	Qld.	Interstate (Surface)
1 - 3	$0.41	$0.41
4 - 18	$0.95	$1.10
19 - 90	$1.90	$2.50
> 90	expensive!	

UTILITIES

Oct.	83	APC	52,4	BASIC program conversion. (Surya)	(2)
Jan.	84	APC	20-21	Beginners tips. (White)	(-)
Nov.	83	APC	57,9	Program conversion Pt. 2 (Surya)	(2)
Nov.	83	APC	89-95	BASIC converter chart. (Surya)	(7)
Feb.	84	APC	140-1	Program conversion Pt. 2 (Surya)	(2)
Mar.	84	APC	42-3	Program conversion - Apple II (Surya)	(2)
Apr.	84	APC	71-2	Program conversion - TRS 80/System 80 (Surya)	(1)
May	84	APC	75-6	Program conversion - Atari (Surya)	(2)
Jun.	84	APC	67	Program conversion - Sinclair (Surya)	(1)
Jul.	84	APC	129-30	Program conversion - BBC (Surya)	(2)
Mar.	84	ETI	63	More functions for the VZ-200. (Olney)	(1)
Apr.	85	ETI	117	Notes and errata for Olney.	(-)
Jul.	84	BB	56	Some more routines. (Middlemiss)	(1)
Jul.	84	M80	3-4	VZED - three new functions.	(1)
Aug.	84	M80	2	VZ-200 output latch.	(1)
Aug.	84	M80	9,15,16	Memory peek VZED. (Carson)	(1)
Aug.	84	M80	3-4	Microsoft ROM BASIC Level I bug.	(1)
Apr.	85	APC	97	VZ-200 bug. (Tritscher)	(-)
Aug.	85	APC	31	VZ bug. (Tritscher)	(-)
Aug.	84	APC	94	VZ-200 moving message and trace. (Batterson)	(1)
Nov.	84	APC	76	Trace function. (Breffit)	(-)
Nov.	84	APC	125	VZ-200 correction. (Kelly)	(-)
Sep.	84	CI	19	VZ200 Input. (Woolf)	(1)
Sep.	84	BB	63	Poking extra functions. (Clark & Hill)	(1)
Oct.	84	ETI	135-7	Extending VZ-200 BASIC. (Olney)	(3)
Nov.	84	APC	125-6	TRON/TROFF function for VZ-200. (Thompson)	(1)
Nov.	84	APC	208-12	MON-200 machine code monitor. (Stamboulidas)	(5)
Nov.	84	PCG	55-56	Lprinter. (Quinn)	(2)
Nov.	84	PCG	suppl.	VZ-200 reverse video.	(1)
Dec.	84	BB	64	Enlarged characters. (Velde)	(1)
Feb.	85	APC	171	BASIC understanding. (Hobson)	(1)
Feb.	85	APC	20	VZ-200 into puberty - Olney's extended BASIC.	(1)
Feb.	85	ARA	19-26	Calculating grey line. (Baker)	(6)
Mar.	85	CI	12-14	Renumber. (Marsden)	(3)
Apr.	85	PCG	62-64	Find. (Stamboulidas)	(3)
Apr.	85	APC	19	Use of RND in dice and card games. (Holland)	(1)
Apr.	85	APC	103	VZ variable definition. (Stamboulidas)	(1)
Apr.	85	APC	95	Variable GO TO on VZ. (Olsen)	(1)
Jul.	85	APC	176	Correction to VZ variable GO TO.	(-)
May	85	APC	52-3	Lysco support for VZ-200. (Young)	(1)
May	85	ETI	99-101	VZ-200 hardware interrupt. (Olney)	(3)
May	85	APC	110	Background VZ. (Williams)	(1)
Aug.	85	APC	130	VZ-200 instant colour. (Willows)	(-)
Aug.	85	APC	130-3	Reversed REM. (Quinn)	(1)
Sep.	85	APC	145	Real-time clock. (Griffin)	(1)
Oct.	85	APC	218	APC benchmark BASIC programs.	(1)
Oct.	85	APC	147	VZ deletions. (Quinn)	(1)
Nov.	85	APC	189	VZ EDITOR/ASSEMBLER tips. (Lam)	(1)
Nov.	85	ETI	94-5	Olney's Level II BASIC for VZ200/300. (Rowe)	(2)

Jan.	86	APC	83,5	VZ user graphics.	(1)
Feb.	86	APC	127	Machine language calls.	(1)
Mar.	86	APC	chart	APC BASIC converter chart 1986.	(8)
Mar.	86	YC	103-5	VZ-200 cassette inlays. (Dutfield)	(3)
May	86	APH	54-55	VZ and photography. (Kohen)	(2)
Jun.	86	APC	209	VZ pause.	(1)
Aug.	86	ETI	86-89	VZ software mods. (CHIP-8 Editor) (Griffin)	(3)
Oct.	86	ETI	28-33	VZ CHIP-8 Interpreter. (Griffin)	(5)
Sep.	86	AEM	89-92	Screen handling on VZ. Part I. (Kitch)	(4)
Oct.	86	AEM	110-112	Screen handling on VZ. Part II. (Kitch)	(4)
Oct.	86	AEM	113,4,21	Reference list of VZ articles. (Kitch)	(2)
Oct.	86	ETI	47	Labeller. (Gallagher)	(1)
Oct.	86	ARA	38-42	Amateur radio logger. (Johnson)	(5)
Nov.	86	EA	35	Speaker enclosure calculator. (Allison)	(1)
Dec.	86	AEM	90-95	Memory mapping on VZ. (Kitch)	(6)
Mar.	87	AR	10-12	Feedline calculations. (Buhre)	(3)
Apr.	87	EA	100-101	Op amp noise. (Allison)	(2)
Apr.	87	ARA	20-24	Beam Headings. (Baker)	(5)
May	87	AEM	86-88	VZ Epson printer patch. (Taylor)	(3)
Jun.	87	AEM	74,75,79	VZ Epson printer patch Pt II.	(3)
Aug.	87	AEM	82-83	VZ expanded EPROM. (Meager)	(2)
-	88	BYC	88	Restore file. (Banks & Saunders)	(1)
-	88	-	-	B-file copier. (Buhre)	(1)
Feb.	88	ETI	70	String file name. (Hand)	(1)
Jul.	88	ETI	74	Disk directory dumper. (Tunny)	(1)
Oct.	88	ETI	124	CTRL-Break disabler. (Tunny)	(1)
Oct.	88	AEM	96-97	VZBUG. (Batger)	(2)
Nov.	88	ETI	120	Clock. (Tunny)	(2)
Feb.	89	ETI	118-119	DOS Hello (Tunny)	(1)
Feb.	89	ETI	119-120	Visisort (Sheppard)	(2)
Nov.	89	ETI	73	Restore (Rowe)	(1)
Nov.	89	ETI	73	Hex/dec conversion (Maunder)	(1)
Jan.	90	CBA	17-19	Beam headings (Baker)	(3)

GAMES

Nov/Dec83	SYN	22-24	Projectile Plotting (Grosjean)	(2)
Dec. 83	APC	161-3	Missile Command. (Whitwell)	(2)
Feb. 84	BB	50-51	Caddy and Reaction Test. (Hartnell)	(2)
Jan. 84	YC	65	Graphic Sine Waves for VZ-200. (Nickasen)	(1)
Apr. 84	APC	178-80	Moon Lander. (Alley)	(2)
Jul. 84	APC	174-8	Blockout. (Pritchard)	(3)
Jul. 84	M80	7,22	Battleships. (Carson)	(1)
Jul. 84	M80	7,20,21	Junior Maths. (Carson)	(2)
Aug. 84	M80	9,16	Contest Log VZED. (Carson)	(1)
Aug. 84	M80	9,16,17	Dog Race VZED. (Carson)	(1)
Oct. 84	PCG	55-7	High Resolution Graphics Plotting. (Thompson)	(3)
Nov. 84	PCG	82	Tips for 'Ladder Challenge', 'Panik' and 'Asteroids'.	(1)
Jan. 85	PCG	54	POKE's to 'Ghost Hunter'.	(-)
- 85	BYC	146-7	Golf Simulation. (McCleary)	(2)
Mar. 86	CFG	4-5	Golf Simulation. (McCleary)	(-)
- 85	BYC	147	Knight's Cross. (Lucas)	(1)
Jan. 85	APC	129-31	Sketcher. (Leon)	(3)
Jan. 85	YC	88-89	Punch. (Rowe)	(2)
Jan. 85	PCG	44-48	Space Station Defender. (Shultz)	(5)
Feb. 85	CI	27-28	Lost. (Potter)	(2)
Mar. 85	YC	105-9	Decoy. (Rowe)	(2)
Mar. 85	CI	-	Mouse Maze. (Crandall)	(1)
Apr. 85	YC	160	Painter. (Daniel)	(1)
Apr. 85	PCG	65-7	Roadrace. (Thompson)	(3)
May 85	YC	106	Number Sequence. (Thompson)	(1)
May/Jun85	PCG	63-7	Sketchpad. (Thompson)	(5)
Jun 85	YC	70	Morse Tutor program. (Heath)	(1)
Jan. 86	YC	150-1	Morse Tutor - again. (Heath)	(2)
Jul. 85	YC	81	Electric Tunnel. (Daniel)	(1)
Aug. 85	YC	114	Number Slide. (Daniel)	(1)
Oct. 85	PCG	47-52	Cube. (McMullan)	(6)
Oct. 85	YC	105-7	Yahtzee. (Thompson)	(3)
Mar. 86	APC	208-9	VZ Frog. (Alley)	(1)
May 86	ETI	93	Balloon Safari, The Drop and Flatten. (Sheppard)	(1)
Jul. 86	YC	75	Simon. (Proctor)	(1)
- 88	BYC	76	Drawing Program. (Winter)	(1)
- 88	BYC	77	Tea-pot Song. (Winter)	(1)
- 88	BYC	78	Ping Tennis. (Duncan)	(1)
- 88	BYC	79-82	Concentration. (Vella)	(4)
- 88	BYC	83	Super Snake Trapper. (Duncan)	(1)
- 88	BYC	84	Worm. (Thompson)	(1)
- 88	BYC	85	Dogfight. (Thompson)	(1)
- 88	BYC	86-87	Bezerk. (Banks & Saunders)	(2)
- 88	BYC	87	Argggggh! (Banks & Saunders)	(1)
- 88	BYC	87	Encode/Decode. (Banks & Saunders)	(1)
- 88	BYC	88	Catch. (Banks & Saunders)	(1)
Apr. 88	ETI	65	U-foe. (Alderton)	(1)
Jul. 88	ETI	73	Disintegrator. (Stibbard)	(1)
Aug. 88	ETI	65	Star Fighter. (Roberts)	(1)
Nov. 88	ETI	121	Drawing Board. (Maunder)	(1)
May 89	ETI	87-88	Camel (Maunder)	(2)

BUSINESS

Aug.	84	APC	172-7	Database VZ-200. (Barker)	(6)
Oct.	84	APC	214	WP for VZ-200. (McQuillan)	(-)
Oct.	85	APC	82-3	Comment on Barker's and Quinn's DB. (Lukes)	(-)
Oct.	84	APC	126-30	Minicalc Spreadsheet. (Stamboulidas)	(5)
Dec.	84	APC	214	Correction to Minicalc.	(1)
May	85	APC	162-3	Micro Type(WP). (Browell)	(2)
Jul.	85	APC	164-6	Database. (Quinn)	(2)
Feb.	88	ETI	72	VZ Wordprocessor. (Tunny)	(1)

PERIPHERALS

Feb.	84	EA	131-2	Real-world interface.	(1)
Aug.	84	EA	65	Improved graphics on VZ-200. (Dimond)	(1)
Aug.	84	PCG	83	I/O card for VZ-200. (ad)	(1)
Oct.	84	APC	214	Serial help request. (Pope)	(1)
Dec.	84	APC	36	Add-ons for VZ-200. (Bleckendorf)	(-)
Oct.	85	YC	140	VZ200/300 Modem. (ad)	(-)
Nov.	84	BI	3,4	RTTY with VZ200. (Keatinge)	(2)
Nov.	84	ETI	106-12	A 'Glass-Teletype' using the VZ-200 Pt I	(7)
Dec.	84	ETI	93-7	" " " " Pt II	(5)
Aug.	85	ETI	72-8	VZ-200 terminal.	(7)
Jun.	86	EA	106	VZ serial terminal. (ad DSE kit K6317)	(-)
				Assembler listing of RS-232 ROM software	(13)
Sep.	85	AR	10-11	Another RTTY. (Butler)	(2)
Jan.	86	AR	19-20	Morse on RTTY. (Butler)	(2)
Feb.	86	ETI	72-4	Modifying VZ-200 16K memory expansion. (Olney)	(3)
Mar.	86	ETI	48	Talking VZ-200. (Bennets)	(1)
Jul.	86	ETI	55-60	Super II VZ-200 hardware modifications. (Sorrell)	(6)
Oct.	86	ETI	14	Errata for Super II.	(-)
Jan.	87	EA	60	EPROM programmer modification. (Buhre)	(1)
Feb.	87	AR	16-17	Morse Interface. (Forster)	(2)
May	87	EA	51	16K Memory Expansion VZ300. (Kosovich)	(3)
Jan.	88	EA	174	VZ-300 expansion problem.	(-)
Aug.	88	EA	138	VZ-300 expansion.	(-)
May	89	EA	124-125	RAM Expansion - Discussion (Sorrell)	(-)
Oct.	88	EA	140	Circuit idea.	(-)
Jun.	87	EA	129	Errata Memory Expansion.	(-)
Jun.	87	AEM	8	VZ software. (Thompson)	(1)
Apr.	88	AR	11-15	Memory expansion for V2200/300	(5)
Apr.	88	AEM	57-63	Ultra-graphics adaptor. (Sorrell)	(8)
Jun.	88	AEM	7	Correction.	(-)
Jul.	88	AEM	7	Correction.	(-)
May	88	ETI	70	VZ amp. (Merrifield)	(1)
Apr.	89	ETI	96	Better VZ amp. (Hobson)	(-)
May	88	ETI	82-86	VZ300 EPROM programmer. (Nacinovich)	(5)
Jun.	88	ETI	86-89	" " " "	(4)
				BASIC listing of software	(5)
Jul.	88	ETI	88-92	VZ300 data logger. (Sutton)	(5)

COMMERCIAL SOFTWARE REVIEWS

Mar.	84	APC	190-1	Review of DSE 'Matchbox', 'Biorhythms', 'Circus' and 'Poker'. (Davies)	(2)
Aug.	84	PCG	46-47	Review of DSE 'Panik' and 'Ladder Challenge'.	(1)
Oct.	84	PCG	90-91	Review of DSE 'Knights and Dragons', 'Ghost Hunter', 'Othello', and 'Invaders'.	(2)
Nov.	84	PCG	90-96	Review of LYSCO 'Cub Scout' and DSE 'Dracula's Castle'.	(1)
Jan.	85	PCG	65	Review of DSE 'Air Traffic Controller' and 'Tennis'.	(1)
Feb.	85	PCG	76	Review of DSE 'Defence Penetrator' and 'Star Blaster'.	(1)
Mar.	85	PCG	76-77	Review of DSE 'Planet Patrol' and 'Learjet'.	(1)
Apr.	85	PCG	94-99	Review of DSE 'Asteroids', Super Snake' and 'Lunar Lander'.	(1)
Apr.	85	ETI	103	Logbook and Morse on VZ-200.	(1)
Oct.	85	PCG	68-9	Review of DSE 'Duel'.	(1)
Nov.	85	PCG	70-1	Review of DSE 'Attack of the Killer Tomatoes'.	(1)
Nov.	85	CLC	31	Review of educational software.	(1)

SOFTWARE ADVERTISEMENTS

A 15 page compilation of ads. for a variety of software, services, User groups etc. (12)

HARDWARE REVIEWS

Apr.	83	YCU	56-59	Texet TX-8000. (Bennett)	(3)
Apr.	83	APC	58-66	VZ-200. (Hartnell)	(5)
Apr.	83	CC	38-43	Review of VZ-200.	(3)
May	83	CC	26-30	Video Technology VZ-200 PC. (Ahl)	(3)
Jun.	83	EA	137	New low-cost computer - VZ-200.	(1)
Jun.	83	ETI	30	Dick Smith colour computer.	(1)
Jun.	83	YC	6	DSE VZ-200.	(-)
Aug.	84	PCG	12	VZ-200.	(-)
Jul.	83	ETI	32-7	DSE's personal colour computer. (Harrison)	(3)
Jul.	83	EA	130-3	The VZ-200: colour, graphics and sound. (Vernon)	(4)
Jul.	83	PCN	16	Timing the Laser's phazer. (Stokes)	(1)
Sep.	83	WM	40	Laser.	(-)
Sep.	83	BB	18-20	Dick Smith VZ200: good value. (Fullerton)	(3)
Aug.	83	YC	20-33	Cash and Carry Computers. (Bell)	(9)
Sep.	83	CC	202-4	Review of VZ-200 and PP40.	(1)
Oct.	83	APC	77-8	VZ-200.	(1)
Oct.	83	WM	135	Texet TX8000.	(1)
Oct.	83	CT	12	The Laser 200.	(-)
Dec.	83	CT	11	Laser 200.	(-)
Nov.	83	CT	37-40	A look at the Laser. (Green)	(4)
Nov.	83	WM	42-108	The Laser - a shot in the dark.	(3)
Nov/Dec	83	SYN	17-22	VZ-200. (Ahl)	(2)
Feb.	84	CC	218-21	Laser PP40 Printer/Plotter.	(2)
Spring	84	MC	52-4	Laser 200. (Green)	(3)
Jun.	84	EA	12-9	Buying your first computer. (Vernon)	(6)
Aug.	84	EA	30-3	An important role for small computers. (Williams)	(4)
Oct.	84	PCG	82-87	Home micro supertest. Pt. 3 (Bollington)	(5)
Nov.	84	PCG	14-19	Home micro supertest. Pt. 4 (Bollington)	(4)
Nov.	84	EA	78-80	VZ-200 as a WP (DSE E&F tape WP). (Williams)	(2)
Dec.	84	CHC	28-31	Review of video games consoles.	(4)
Mar.	85	EA	31-33	Back to the VZ-200. (Williams)	(1)
Jul.	85	ETI	102-6	Dick Smith's new VZ-300. (Rowe)	(5)
Aug.	85	EA	22-7	WP on the new VZ-300. (Williams)	(5)
Dec/Jan	86	PCG	11-15	How to buy a micro - VZ-300 compared.	(4)
Aug.	86	AHC	38-39	Computers for the Rest of Us. (Roberts)	(2)
Nov.	86	AHC	44	Letter. (Kennedy)	(-)
Dec.	87	YC	20-21	VZ-300. (Hartnell)	(2)
Dec.	87	YC	78	VZ-300	(1)

GENERAL PROGRAMMING

Apr.	81	ETI	87-93	Extra Z80 opcodes.		(4)
Jun.	81	ETI	97	More uncovering Z80. (Dennis)		(1)
Jul.	81	ETI	83	Z80 uncovered. (Garland)		(-)
				Z80 CPU reference card		(2)
Feb.	82	YC	64-66	Understanding Assembler (Bell)	Part I	(3)
Mar.	82	YC	74-77	(8080)	Part II	(4)
Apr.	82	YC	61-63	" " "	Part III	(3)
May	82	YC	60-62	" " "	Part IV	(3)
Jun.	82	YC	99-101	" " "	Part V	(3)
Jul.	82	YC	1-74	" " "	Part VI	(3)
Sep.	82	YC	57-59	" " "	Part VII	(3)
Nov.	82	YC	45-46	" " "	Part VIII	(2)
Dec.	82	YC	93-97	" " "	Part IX	(4)
Jan/Feb	83	YC	52-55	" " "	Part X	(4)
Mar.	83	YC	61-62	" " "	Part XI	(2)
Aug.	83	YC	62-68	" " "	Part XII	(6)
Oct.	83	YC	87-89	" " "	Part XIII	(2)
Nov.	83	YC	102-104	" " "	Part XIV	(3)
Feb.	84	YC	93-94	" " "	Part XV	(2)
Apr.	84	YC	123-126	" " "	Part XVI	(2)
Nov.	82	PE	1/1-1/5	PE Micro-file #1 - 8080 & 8085 (Coles)		(5)
Jan.	83	PE	3/1-3/5	PE Micro-file #3 - Z80. (Coles)		(5)
Mar.	84	APC	73-85	Teach yourself assembler	Pt. 1 (Overaa)	(6)
Apr.	84	APC	57-64	(8080, Z80, 6502)	Pt. 2 (Overaa)	(5)
May	84	APC	89-98	" "	Pt. 3 (Overaa)	(5)
Jun.	84	APC	53-60	" "	Pt. 4 (Overaa)	(5)
Jul.	84	APC	61-64	" "	Pt. 5 (Overaa)	(3)
Aug.	84	APC	110-116	" "	Pt. 6 (Overaa)	(5)
Sep.	84	APC	145-151	" "	Pt. 7 (Overaa)	(4)
Jan.	85	APC	122-124	Sort at input. (Ithell)		(1)
Feb.	85	APC	103-109	The basic art - algorithms, structures. (Liardet)		(4)
Mar.	85	APC	98-109	Pick a number - arithmetic. (Liardet)		(5)
Apr.	85	APC	79-87	It takes all sorts - sorting. (Liardet)		(5)
Oct.	85	APC	82	The Art of Programming - Progress. (Hjaltson)		(-)
Jun.	85	APC	170-171	Comment on binary search. (Lamich)		(1)
Jun.	85	APC	171-173	Comment on distribution sort. (Riordon)		(1)
Oct.	85	YC	107-8	Sorting out the sorts. (Jankowski)		(1)
Mar.	86	PE	17-18	Z80		(2)

AEM	Australian Electronics Monthly	ETI	Electronics Today International
AHC	Australian Home Computers		
APC	Australian Personal Computer	M80	Micro-80
APH	Australian Photography		
AR	Amateur Radio		
ARA	Amateur Radio Action		
BB	Bits and Bytes (NZ)		
BI	Break In (NZ)		
BYC	Bumper Book of Programs by YC	MC	Micro Choice (UK)
CBA	CB Action		
CC	Creative Computing (US)	PCG	Personal Computer Games
CFG	Computer Fun and Games	PCN	Personal Computer News (UK)
CI	Computer Input (NZ)	PE	Practical Electronics (UK)
CLC	Classroom Computing	SYN	Sync (US)
CT	Computing Today (UK)	WM	Which Micro (UK)
CHC	Choice	YC	Your Computer
EA	Electronics Australia	YCU	Your Computer (UK)

FURTHER LITERATURE RELATING TO THE VZ200/300 COMPUTER

As an extension to my list of magazine articles, I have produced the following list of books (I have copies of all of the publications). The books relate to the VZ computer specifically, Microsoft BASIC Level II or the Z-80 microprocessors, as used in the VZ200/300. Additionally, I hold a lot of additional technical information, ROM listings, Users Group newsletters, software etc.

TECHNICAL BULLETINS FOR VZ COMPUTERS

# 88	Printing out System-80 screen graphics.	(2)
# 91	Programming the VZ-200 computer's joysticks.	(3)
# 92	Finding where variables are stored by the VZ-200's BASIC.	(3)
# 93	Problems with the X-7208 printer/plotter and Microsoft BASIC.	(1)
# 94	Using the X-3245 TP-40 printer/plotter with the VZ-200 & System-80.	(1)
# 98	Printing lower case and control characters on the VZ200/300.	(1)
#111	VZ-300 Mailing List tape to disk file conversions.	(1)
#114	Obtaining colour on the VZ300.	(1)
#116	Fixing the printer bug in the VZ Editor-Assembler.	(1)
	Letter on tapes and keyboard	(1)
	General hints on VZ	(1)
	Service Manual for printer interface	(7)
	Service Manual for disk drive controller	(12)

BOOKS ON VZ COMPUTERS

Henson, T.L.,	1983	"Introduction to Computing". DSE, 114 p.	(60)
Hartnell, T., & Predebon, N.,	1983	"Getting Started". DSE, 121 p.	(68)
Hartnell, T.,	1983	"Further Programming". DSE, 135 p.	(74)
Hartnell, T., & Pringle, G.,	1983	"The Giant Book of Games". DSE, 179 p.	(94)
-	1983	"First Book of Programs". DSE, 58 p.	(60)
-	1983	"Second Book of Programs". DSE, 57 p.	(60)
Rowe, J.,	1983	"VZ-200 Technical Reference Manual". DSE, 22 p.	(30)
-	1985	"VZ-300 Technical Manual". DSE, 39 p. (Available from DSE $14.95)	(65)
Hartnell, T.,	1986	"Programming the VZ300". DSE, 171 p. (Available from DSE $14.95)	
Hartnell, T.,	1986	"The Giant Book of Games for the VZ300". DSE, 278 p. (Available from DSE $19.95)	
Hartnell, T.,	1986	"The Amazing VZ300 Omnibus". DSE, 188 p. (Available from DSE $19.95)	
Wolf, G.,	1985	"ROM-listings fur Laser 110, 210, 310 und VZ200". Vogel-Buchverlag. 278 p.	
Wolf, G.,	1985	"Der BASIC-Interpreter in Laser 110, 210, 310 und VZ200". Vogel-Buchverlag. 152 p.	
Wolf, G.,	1985	"Das Laser-DOS fur Laser 110, 210, 310 und VZ200". Vogel-Buchverlag. 131 p.	
Sanyo,	1984	"Mein Laser Home-Computer, Tips and Tricks fur Einsteiger". Sanyo Video Vertrieb. 91 p.	
Sanyo,	1984	"Laser Home-Computer, Software-System Handbuch I". Sanyo Video Vertrieb. 114 p.	
D'Alton, J.,	1986	"Vprogrammez Hints and Hardware No. 1" 48 p.	
Schaper, P.,	1987	"Beginners Guide to the VZ 200/300 Editor Assembler" 57 p.	
Olney, S.	1987	"VZ 200/300 Assembly Language Programming Manual for Beginners". 140 p.	

BOOKS ON BASIC

Albrecht, R.L., Finkel, L., & Brown, J.R.,	1978	"BASIC". John Wiley, 2nd Edition. 325 p.
Albrecht, B., Inman, D., & Zamora, R.,	1980	"TRS-80 BASIC". John Wiley. 351 p.
Inman, D., Zamora, R., & Albrecht, B.,	1981 1981	"More TRS-80 BASIC". John Wiley. 280 p.
Lien, D.A.,	1982	"Learning TRS-80 BASIC". Compusoft. 528 p.
Gratzer, G.A. & Gratzer, T.G.,	1982	"Fast Basic - beyond TRS-80 BASIC". John Wiley. 278 p.
Rosenfelder, L.,	1981	"BASIC Faster and Better and other mysteries". IJG, California. 288 p.
Bardon, W.,	1985	"TRS-80 Computer Reference Handbook" Radio Shack 2nd edit.

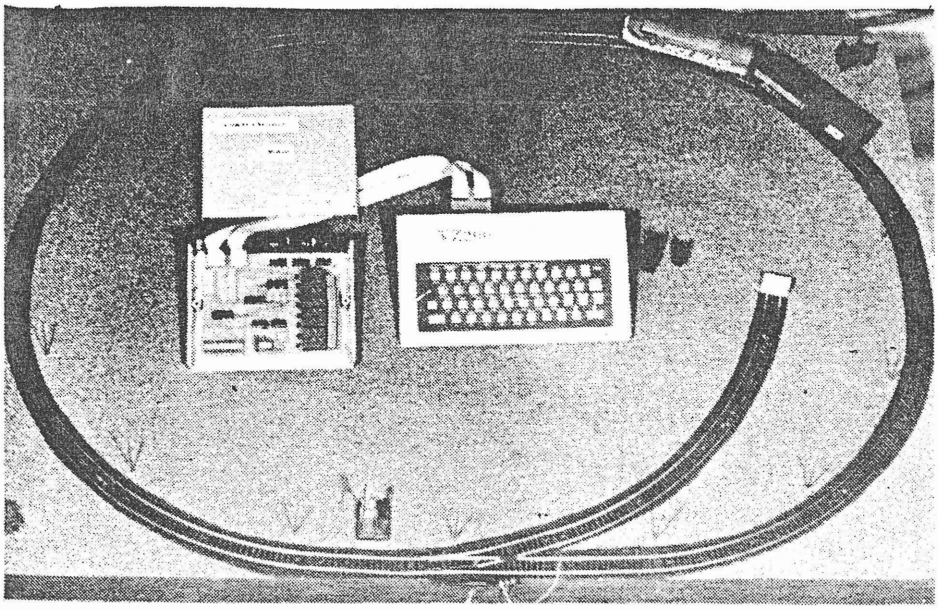
BOOKS ON ASSEMBLER AND Z80

Carr, J.J.,	1980	"Z80 Users Manual". Reston Publishing Co., 326 p.
Weller, W.J.,	1978	"Practical Microcomputer Programming: the Z80". Northern Technology, 481 p.
Fernandez, J.N., & Ashley, R.	1981	"Introduction to 8080/8085 Assembly Language Programming". John Wiley, 303 p.
Miller, A.R.,	1981	"8080/Z80 Assembly Language - techniques for improved programming". John Wiley, 318 p.
Leventhal, L.A.,	1979	"Z80 Assembly Language Programming". Osborne/McGraw-Hill.
Leventhal, L.A., & Saville, W.	1983	"Z80 Assembly Language Subroutines". Osborne/McGraw-Hill, 497 p.
Nitschke, W.,	1985	"Advanced Z80 - Machine Code Programming". Interface Publications, 342 p.

Nichols, J.C., Nichols, E.A., & Rony, P.R.	1979	"Z-80 microprocessor programming and interfacing - Book 1". Howard W. Sams, 302 p.
Nichols, J.C., Nichols, E.A., & Rony, P.R.	1979	"Z-80 microprocessor programming and interfacing - Book 2". Howard W. Sams, 494 p.
Nichols, J.C., Nichols, E.A., & Musson, K.R.	1983	"Z-80 microprocessor advanced interfacing with applications in data communications". Howard W. Sams, 347 p.
Barden, W.,	1979	"TRS-80 Assembly-Language Programming". Radio Shack, 224 p.
Barden, W.,	1982	"More TRS-80 Assembly-Language Programming". Radio Shack, 430 p.
Farvour, J.L.		"Microsoft BASIC Decoded and other mysteries". IJG, California, 310 p.
Sargent, M., & Shoemaker, R.L.	1981	"Interfacing Z80 microcomputers to the real world". Addison Wesley, 288 p.
Ullman, J.,	1984	"Pocket Guide Assembly Language for the Z80". Pitman, 58 p.
Overea, P.A.,	1984	"Teach Yourself Assembler Z80". Century Communications, London, 236 p.
Barrow, D.,	1985	"Assembler Routines for the Z-80". Century Communications, London, 192 p.
Uffenbeck, J.,	1985	"Microcomputers and Microprocessors: the 8080, 8085 and Z80. Programming, Interfacing and Troubleshooting". Prentice Hall, 670 p.
Barden, W.,	1978	"The Z80 Microcomputer Handbook" Howard Sams, 304 p.
Goodwin, M.	1983	"Level II ROMS" Tab Books, 536 p.
Blattner, J., & Mumford, B.,	1980	"Inside Level II" Mumford Micro Systems, 65 p.
Barden, W.,	1982	"TRS-80 Assembly Language Subroutines" Prentice Hall, 232 p.
Toothill, A., & Barrow, D.,	1983	"Z80 Code for Humans" Granada, 152 p.

PERIPHERALS

Feb.	84	EA	131-2	Real-world interface.	(1)
Aug.	84	EA	65	Improved graphics on VZ-200. (Dimond)	(1)
Aug.	84	PCG	83	I/O card for VZ-200. (ad)	(1)
Oct.	84	APC	214	Serial help request. (Pope)	(1)
Dec.	84	APC	36	Add-ons for VZ-200. (Bleckendorf)	(-)
Oct.	85	YC	140	VZ200/300 Modem. (ad)	(-)
Nov.	84	BI	3,4	RTTY with VZ200. (Keatinge)	(2)
Nov.	84	ETI	106-12	A 'Glass-Teletype' using the VZ-200 Pt I	(7)
Dec.	84	ETI	93-7	" " " " Pt II	(5)
Aug.	85	ETI	72-8	VZ-200 terminal.	(7)
Jun.	86	EA	106	VZ serial terminal. (ad DSE kit K6317)	(-)
				Assembler listing of RS-232 ROM software	(13)
Sep.	85	AR	10-11	Another RTTY. (Butler)	(2)
Jan.	86	AR	19-20	Morse on RTTY. (Butler)	(2)
Feb.	86	ETI	72-4	Modifying VZ-200 16K memory expansion. (Olney)	(3)
Mar.	86	ETI	48	Talking VZ-200. (Bennets)	(1)
Jul.	86	ETI	55-60	Super II VZ-200 hardware modifications. (Sorrell)	(6)
Oct.	86	ETI	14	Errata for Super II.	(-)
Jan.	87	EA	60	EPROM programmer modification. (Buhre)	(1)
Feb.	87	AR	16-17	Morse Interface. (Forster)	(2)
May	87	EA	51	16K Memory Expansion VZ300. (Kosovich)	(3)
Jan.	88	EA	174	VZ-300 expansion problem.	(-)
Aug.	88	EA	138	VZ-300 expansion.	(-)
May	89	EA	124-125	RAM Expansion - Discussion (Sorrell)	(-)
Oct.	88	EA	140	Circuit idea.	(-)
Jun.	87	EA	129	Errata Memory Expansion.	(-)
Jun.	87	AEM	8	VZ software. (Thompson)	(1)
Apr.	88	AR	11-15	Memory expansion for V2200/300	(5)
Apr.	88	AEM	57-63	Ultra-graphics adaptor. (Sorrell)	(8)
Jun.	88	AEM	7	Correction.	(-)
Jul.	88	AEM	7	Correction.	(-)
May	88	ETI	70	VZ amp. (Merrifield)	(1)
Apr.	89	ETI	96	Better VZ amp. (Hobson)	(-)
May	88	ETI	82-86	VZ300 EPROM programmer. (Nacinovich)	(5)
Jun.	88	ETI	86-89	" " " "	(4)
				BASIC listing of software	(5)
Jul.	88	ETI	88-92	VZ300 data logger. (Sutton)	(5)



Real-world interface suits any computer

Sydney firm Meyertronix now has available a computer input/output unit suitable for use with any Z80-based computer system. The unit, available either as a kit or fully assembled, provides eight digital inputs, eight outputs to relays and a single programmable analog voltage output.

The unit we have seen came complete with cables and connectors for the VZ-200 computer, but interfacing requires only the connection of four address lines, the data bus and the Z80 control signals IORQ, RD and WR, making it suitable for the ZX81, MicroBee and Super 80 computers, among others. A version is also available for the Commodore 64 and VIC 20 computers.

If more than eight inputs and outputs are required, up to five boards can be connected in parallel, using a special cable arrangement.

The unit is supplied in an ABS plastic case measuring 196 x 158 x 64mm. The main circuit board measures 170 x 133mm and is double-sided with plated-through holes. Eight ICs are used, with data and address line connections to and from the board made by DIP header sockets. The VZ-200 version also comes with a smaller PCB terminated in a 30-way edge connector suited to the peripheral interface of the computer. Power for the circuitry is provided from the computer itself.

Address decoding is performed on-board, with three locations allocated — one each for the eight bit input and output ports and a separate port for the analog voltage output. The decoding is hard-wired, so that the port addressing cannot easily be changed. In the Z80 version the input port is at location 80 hex (128 decimal), the relay output port at 81 (hex) and the analog output port at 82 hex.

The method of producing the analog voltage is interesting. One eight bit output port is dedicated to this function and drives a set of eight analog switches. These switches in turn connect one or more resistors in series with the ADJ input of an LM317 adjustable voltage regulator.

Sending a binary code to the output port thus produces a voltage which is adjustable between 1.2V (the minimum output of the LM317) and the maximum input voltage to the regulator (which can be up to 30V if required). Provided that the resistors in the controlling network are selected for precise values, the output is programmable in 256 equal steps.

Programming the controller is simple as the Basic statements OUT and INP do all the work (PEEK and POKE for the Commodore machines). Some trial and error would be required to develop a program capable of close control of the analog voltage output as the relationship between data values and output voltages depends naturally enough on the maximum value of the voltage input to the LM317.

The eight input lines are unlatched and are normally held high by pull-up resistors. Pushbuttons, reed switches or more complex sensors are easily connected and must be arranged so that they pull the appropriate input line to ground when operated. Reading the status of the switches is simply a matter of performing an INP or PEEK statement.

The second output port controls relays which are claimed to be suitable for switching 240VAC at up to 2A. Unfortunately the provision for connecting to the relays is rudimentary, consisting of a terminal block mounted on the PCB inside the case of the unit. The user must supply and run cables to the terminal block, which would require cutting access holes in the case.

The relays are operated by binary codes which of course are represented by decimal values in Basic, but the scheme is easy to use.

Documentation for the unit consists of seven pages of description, construction and application notes, some example software, circuit diagram and PCB overlay. Cost of the unit in kit form is $98, and fully assembled and tested versions are available for $158.

Meyertronix also has available an industrial version of the controller, again designed to interface with any computer system. This version is supplied in modular form in a 19" rack mount cabinet with separate boards each providing eight optically-isolated digital inputs or outputs. A real-time clock, parallel printer interface and parallel printer interface boards are also available.

To use this system a separate address decoder board is required which supplies 128 individual I/O select signals. A power supply board is also required, bringing the cost of a minimum system to around the $1000 mark (depending on the number of input and output boards making up the system).

For further information on either version of the I/O controller contact Meyertronix, PO Box 65, Riverstone, NSW, 2765. Phone (02) 627 2510.

ELECTRONICS Australia, February, 1984

P 131-2.
1 of 1.

Improved graphics for VZ200 Computer

Want a 50% improvement to the graphics resolution of your VZ200 computer? Here's how you do it.

Inside the VZ200 is Motorola's MC6847 video display generator (VDG) chip. This is an easily programmed yet highly versatile device offering several text, mixed text/graphic and graphic modes.

As standard, the VZ200 comes with a 128 × 64 dot 4-colour graphic mode. In this mode, each display byte is split into four dots, each occupying two bits. The two bits specify which of four colours the dot is in one of two 4-colour sets — making eight colours available in all.

This requires all 2K of video RAM.

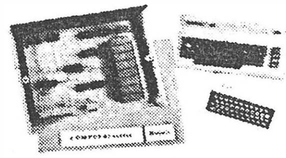
By cutting the track linking pin 30 of U15 (the VDG) to ground and then connecting this pin to +5V (pin 17), a new graphics mode is derived. This mode offers 128 × 96 dot monochrome (ie, two-colour) graphics, where each bit specifies one dot and requires 1.5K of RAM. The advantage of this mode is that the dots are square which improves plot appearance.

A single-pole 2-position switch can be used to switch between one mode and the other. This switch can be mounted on the side of the case.

P. Dimond,
Lidcombe, NSW. **$10**

NEW!

Use Your Computer to Control Real World Equipment!



only

$159.00

Tax inc.

$115 Kit

Does not include

COMPUTAControl Module

For the Dick Smith VZ200, Commodore 64 and all other Z80 based computers.

Applications:

• Sprinkler control in garden or nursery • Memory mapable • Model train control • Slot car monitor • Automate simple machines and processes • Control robots

Available from

bankcard welcome here Phone or mail

MEYERTRONIX

P.O. Box 65, Riverstone, 2765.
Ph.: (02) 627 2510.

PCG. Aug 84 p 83.

Serial help request

Can any of your readers help me with a slight problem that I have encountered. The problem being that nowhere can I find anybody who would be able to tell me how to connect an RS232 to my VZ-200, enabling me to run a modem from the computer.

The VZ-200 is an inexpensive, great little micro but is severely restricted by its lack of ability to connect peripherals such as the modem, which would open up the way for better communications for users.

I would be grateful if any one who has had success, or even ideas on how this can be done, could write to me at 'C/- 187 Port Road, Hindmarsh 5007', or send them into the magazine.

Nigel Pope

APC 5(10) Oct 84 p. 214.

Add-ons for VZ-200

In reply to Nigel Pope's letter in the October issue of *APC:* an RS232 interface for the VZ-200 is available from Mr Ronald Rohde, 13/12 Walsh Street, South Yarra, Vic, 3141. It is sold for $49.95 by mail order only.

Mr Rohde also manufactures and sells various add-ons for the VZ-200, and offers an extensive range of software on cassettes.

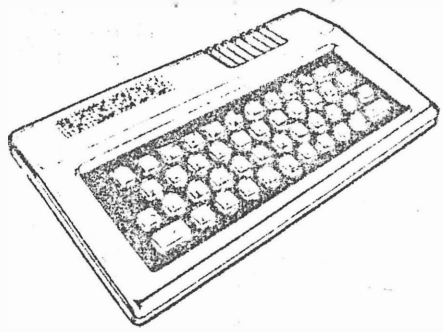
I am a primary school teacher at a Perth primary school using the VZ-200 as an educational tool in classes.

R Bleckendorf

APC 5(12) Dec 84 p 36.

VZ200/300 Modem
RS232 interface with software in ROM. Modem supports Bell 103/ CCIT V.21 300 bps with auto-answer and telephone handset. Phone (03) 791 5850 ah.

YC Oct 85 p 140



Radio Teletype with the Dick Smith VZ-200 Microcomputer

by ROSS KEATINGE, ZL1BNV

About a year ago I was contemplating the purchase of a second microcomputer for the ham shack to be dedicated to radio activities and leaving the present machine (a System-80) available for general computing and development purposes. The requirements were for something compact but with a useable keyboard, the facility to use machine code if required and most important for radio activities, freedom from RF noise generation. This latter point had been a problem with the System-80 on what was) otherwise a good all-round machine. The VZ-200 had just been released and seemed to fit the bill nicely. Bringing one home on trial brought the pleasant surprise of using a machine which was spectrally inaudible when running beside an HF receiver.

After getting over the novelty of being able to draw coloured lines on a television, it was time to get down to the job of getting it going for its intended use, namely RTTY. The first problem was to decide on a method of getting the RTTY signals in and out of the computer. One method considered was to use the expansion bus connection and build a serial I/O port using a UART chip for the parallel/serial conversion. The advantage of this m hod was that much of the software could then be written in BASIC using a simple INP or OUT instruction to send data to, or get data from, the serial port. The disadvantages were the possiblity of RF noise due to the bus being extended outside the computer case and the extra circuitry needed, especially if a variety of baud rates were required. Previous experience led me away from this option. The other possibility was to use the cassette port and machine code software to produce the serial signals. This method was adopted and several advantages became apparent. No expensive edge connector was required, baud rates could be changed easily in software and no other functions of the computer were affected. The circuitry required for this type of interface is considerably simpler than the "standard" type of interface using UARTs with their associated baud rate generators, etc. The interface described here has only three integrated circuits.

Having decided on this method, another problem emerged. There was a requirement for an efficient method of writing machine code for the VZ-200 for which there is no assembler available. The trusty System-80 was pushed into use along with the Microsoft Editor/Assembler package. Software was developed to enable the VZ-200 to load machine code tapes produced by the System-80 and to convert these into VZ-200 format.

The output side of the VZ-200 cassette port is DC coupled with about 200 mV output when programmed high and close to ground when low. A simple comparator is therefore all that is required to convert this to a standard logic signal. The input side is AC coupled so therefore cannot be used directly to detect a logic level. It can, however, be used to detect whether or not an audio tone is present. Therefore by using the logic signal to gate an audio tone on and off, software could read the state of the logic. This method is easily implemented because the audio tone from the AFSK generator can be used for this purpose.

After some experimentation, the circuit of Figure 1 was devised. The XR-2211 was used as a simple and effective means of converting the received mark and space tones to logic levels. The XR-2206 was chosen as the AFSK generator, the output of which is also used in the receive process described above. A Tx/Rx control line

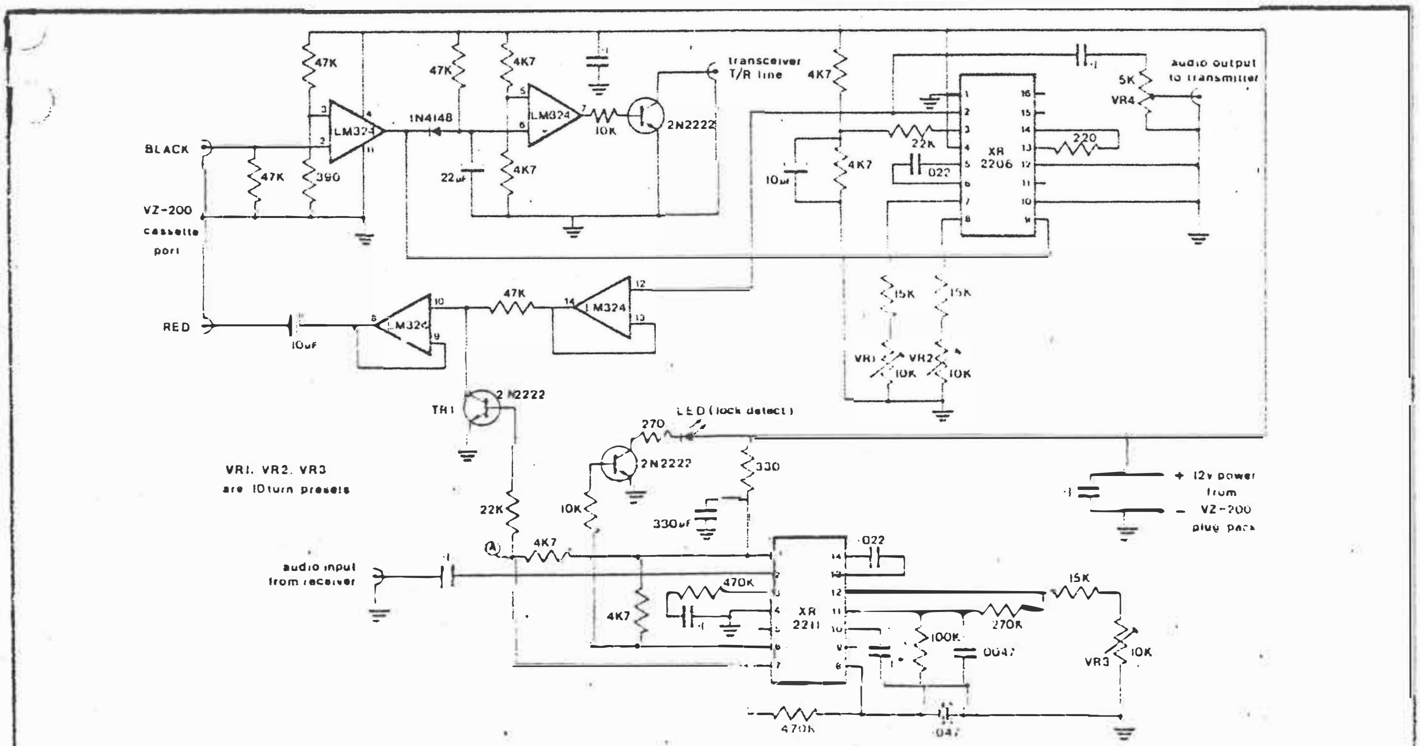


Figure 1 — RTTY AFSK erator and receiver.

for the transceiver is derived from a monostable which switches to transmit when a space condition is sent from the computer and times out after about half a second of constant mark signal. The component values given here have been calculated for the standard amateur tone frequencies of 2125 Hz mark and 2295 Hz space. Those who wish to experiment further are referred to the data on the XR-2211 and XR-2206 available from the agents (Professional Electronics Ltd).

A considerable number of hours were spent developing and modifying the software which in its final form consists of about 1300 lines of Z80 assembler code. It produces about 2.5 kilobytes of machine code when loaded into the VZ-200. The program incorporates the following features:

—split screen display for transmit and receive.
—fully buffered keyboard with 1000 character buffer.
—nine message memories which can be saved on cassette along with the program.
—baud rate keyboard selectable from 45 to 99 baud.
—ability to type in transmit text while still receiving.
—selectable line length on transmit with no breaking of transmitted words.
—both transmit and receive text able to be sent to line printer.
—runs on a standard machine without extra memory.

These features have been selected as being the most useful of the wide range of possibilities available. Users have found the system to be very "friendly" and as good as most commercial packages available for other machines.

Since the program source code cannot be entered and used on a VZ-200, it is not reproduced here. The author will make the machine code available in the form of a standard VZ-200 cassette (see details below).

Construction and Adjustment

The only important construction detail is that the circuit should be built in a grounded metal case to prevent RF from the station transmitter causing problems. The circuit can be simply constructed on copper strip matrix board. Perhaps someone with a flair for artwork will come up with a printed circuit board. The power for the interface can be obtained from the VZ-200 plug pack.

Adjustment is a simple matter of setting the frequencies of the PLL decoder and the AFSK tone generator. To do this a program was written to make the VZ-200 behave like a frequency counter. This is included on the tape containing the main program. The following steps should be followed:

1 Disconnect the collector of TR1. This ensures that the computer is receiving the audio tones from the XR-2206.
2 Load and run the frequency counter program then connect the cassette cables from the VZ-200 to the appropriate connectors on the interface.
3 At this stage the screen should be showing the mark frequency.
4 Adjust VR1 for a frequency of 2210 Hz. This is halfway between the standard frequencies of 2125 Hz and 2295 Hz.
5 Connect the interface input into its audio output. Adjust VR3 for the centre of its lock range as indicated by the lock detect LED.
6 Now adjust VR1 for 2125 Hz, this sets the mark frequency.
7 Press S. The computer now shows the space frequency. Adjust VR2 for a frequency of 2295 Hz.
8 Reconnect the collector of TR1.

This completes the calibration process.

Operation

The XR-2211 works with an input level of between 2 mV and 3 V RMS. If your receiver does not have a low level audio output, a suitable signal can usually be obtained from the top of the AF gain control. Alternatively, the speaker signal can be used but this has the disadvantage of being dependent on the AF gain control. Trimpot VR4 adjusts the audio output level of the interface. A maximum of about 2 V RMS is available. Remember to stay within the continuous power limitations of your transceiver.

The system has been in use for several months now and has given good results on both HF and 144 MHz FM. The operation on HF is achieved by transmitting the audio output of the interface on lower sideband, producing normal FSK. If the transceiver has a direct FSK input available, then this could be driven from the logic signal of pin 1 of IC1. If this is done, remember that the audio tone from the XR-2206 is still required in the receive circuitry. The PLL decoder will decode weak signals well but can be affected by strong interfering signals within the passband. The IF shift on some transceivers can be used to good advantage to reduce interference. If a good quality FSK decoder is already available, it could be used by applying its logic output to TR1 at point A instead of the XR-2211 signal. It should produce a logic high (i.e., turn TR1 on) when the low frequency mark tone is detected.

Overall it has been an interesting project and has enabled several people to enjoy another facet of our hobby without great expense. **See you on the screen!**

A cassette containing both the main RTTY program and the frequency counter program together with a five page instruction booklet can be obtained from the author at a cost of $15.00 (including p & p).

Ross Keating ZL1BNV
1/13 Minto Road,
Remuera,
Auckland 5.

2 of 2.

As radioteletype (RTTY) is an increasingly popular transmission mode amongst radio amateurs, and as we've done a few RTTY projects in the past, we thought this project was a suitable addition to the series. Designed and developed by the R&D Department of Dick Smith Electronics, it is simply an add-on for their popular low-cost VZ200 home computer. Just attach your transceiver and type "CQ DX"!

Neat and simple. The project just plugs into the back of the VZ200. It must be the 'Mini Moke' of modems!

A 'GLASS TELETYPE' USING THE VZ200

IF YOU'RE considering venturing into the world of radioteletype, an ancient and venerable form of digital communications (comparatively speaking), but would like to take the modern route — which means employing a computer — then this project is ideal. Or, if you've been playing with RTTY for some time, but have a combination of the older electromechanical technology and earlier electronic interfaces, and want to update, then this project represents a good 'stepping stone'.

If you're entirely new to radioteletype, then we recommend *"Radioteletype: It's finger-lickin' good"*, in the October '84 issue.

The system

The Dick Smith VZ200 is a low-cost home computer but not lacking in features. One useful feature is a full expansion buss accessible via an edge connector on the main pc board, projecting through the rear of the case. Using this buss, one can attach a variety of peripherals and communicate in and out of the computer by decoding any of the Z80 CPU's ports suitable for the purpose. This project makes use of that facility.

One of the lesser-known features of the VZ200 is its internal RF radiation shielding. If you've ever had an HF receiver near a computer, you'll know just how much and how strong is the 'crud' they radiate from one end of the spectrum to the other!

The VZ200 tackles this computer quirk with the inclusion of extensive tinplate shielding over sections of the circuitry prone to radiation — particularly the memory circuitry. Hence the VZ200 can be sited near sensitive HF receiving equipment without the problems that plague many other computers. It's not entirely free from 'birdies' but, in general, they're out of harm's way. The VZ200 RTTY adaptor was developed by Ian Lindquist, VK2CA and Rex Callaghan, both of Dick Smith Electronics.

The project itself comprises two boards housed in a plastic peripheral box made by the VZ200 manufacturer. One board is the 'decoder' board, which contains the port decoding and RTTY terminal software in an EPROM, while the other board is the modulator/demodulator (or modem) board, containing the tone generator for driving the transmitter and the receiver converter for converting the incoming audio from the receiver and turning it into pulses for the computer to work on.

The idea is that the VZ200's keyboard becomes your erstwhile 'teletype' keyboard, and the video screen becomes your 'printout' — hence the term 'glass teletype'. A printer can be attached to the VZ200's printer port to give you 'hard copy' on paper, if you so desire.

The receiving converter features two cascaded active bandpass filters. These have a steeply rolling-off response to reduce noise and interference; their adjacent 'skirts' coincide, providing an essentially 'flat' bandpass response across the 2100 Hz to 2300 Hz band, neatly enclosing the 'amateur standard' 2125/2295 Hz tones (170 Hz shift) with a little leeway to cope with variations. An XR2211 phase-locked loop is used to generate 'mark' and 'space' pulses from the incoming tones. This chip conveniently provides a 'lock detect' output pin and this is used to drive a LED which lights when you have a signal correctly tuned.

There is one special point worth noting about the PLL. The main VCO frequency determining component is C10, a 22n/400 V metallised polyester capacitor. This was chosen because it has a low temperature coefficient of capacitance around normal room temperatures (25° C). Substitutions may cause problems with excessive temperature drift and uncertain operation.

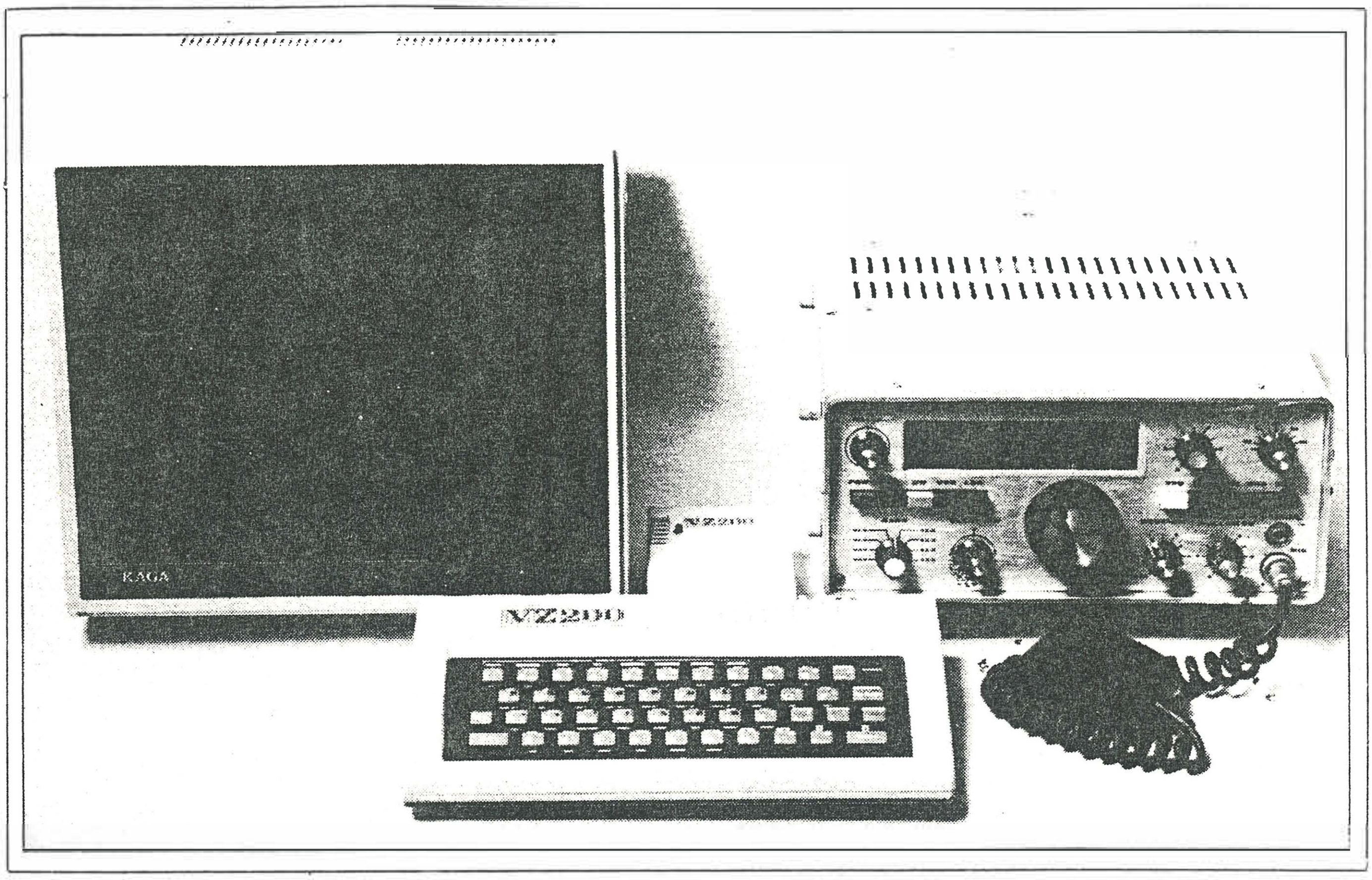
The transmitter section comprises a simple but reliable 'Walsh Function' pseudo-sinewave generator that generates, digitally, the two tones. This is followed by a filter, the output of which is fed to your transceiver's mic input.

Relay control of your transmitter is effected by a relay on the decoder board, the contacts of which go to the push-to-talk contacts (PTT) on your transceiver. This relay, and the transmitter section of the modem board, are each controlled by one of the decoded computer ports.

The project is powered from the VZ200 supply rail, via the expansion connector. The only interconnection required is to your transceiver's mic input, the PTT input and the audio output.

The software provides you with the two 'screens'. The upper screen is used to dis-

1 of 7.



play the text you type, while the lower screen displays the received text. Each screen has independent scrolling. You can type and receive simultaneously. In other words, you can begin typing a reply while receiving text from another station.

You have a 'type ahead' buffer which can contain up to 1024 characters (1K). Apart from that, the software gives you a total of six transmit buffers, one of which is reserved as a 'who are you?' (or WRU) buffer. This versatile feature alerts you when another station calls you by your callsign or some other identification, and the unit will send a response. For example: say VK2ETI wishes to activate your WRU mode. He would send

VK2XYZ WRU VK2ETI

and your unit would respond with something like

STATION IDENTIFICATION
DE VK2XYZ (PETER)

and, if you had put a message in the WRU buffer, your unit could add

STAND BY
++ OPERATOR ALERTED ++

or whatever you had inserted. It is considered impolite to insert messages in the WRU buffer like

RACK OFF HAIRY LEGS!

There are various ways of using this feature, explained later.

There are seven pre-programmed messages stored in the unit's EPROM. Many are designed to insert your callsign automatically when called, saving you time and effort. You can send a string of CQs along with your callsign; a row of RYs (the accepted 'test' signal'; it contains the highest data density); the 'quick brown fox' message along with the numerals 0 to 9 (full alphanumeric series); the 'send — over' terminator; station identification; send your callsign; and send DE followed by your callsign.

There is a total of fourteen 'transmit' commands and nine 'immediate' commands, all called using the SHIFT key. The immediate commands control the overall operation of the 'glass teletype'. One toggles the current mode — i.e: from transmit to receive or from receive to transmit; one exits from the current operating mode to the menu; one controls the WRU mode; one gives you backspace; one changes the baud rate; one returns you to the 'callsign entry' — a sort of 'begin again' command, and two control the printer operation.

Construction

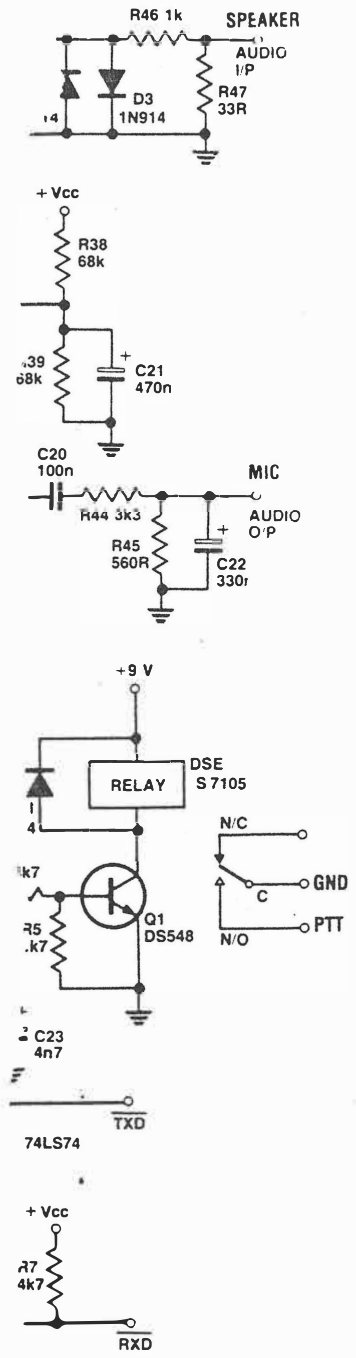
Before commencing any of the electronic assembly, carefully check the track side of each pc board. See that all the holes are drilled and of the correct size. Check that there are no solder 'bridges' between closely-spaced tracks, particularly between IC pads. See that there are no obvious breaks in any tracks.

Probably the best place to start is with the case. It comes in two halves. Mark out positions for the DIN socket and the LOCK DETECT indicator LED on the case lid (the larger piece). See the accompanying photograph. Drill them to size and then insert the DIN socket and screw it in place. The LED mounts on the pc board on the ends of its leads and protrudes through the hole in the case lid. The length of its leads will permit some variation in the exact hole position in the case lid.

Once that's out of the way, you can tackle the board assembly. It's easiest to start with the decoder board. It's marked ETI-756a/ZA1694. There are eight links required on this board; install them first. Use 22g tinned copper wire. Next, install the resistors and capacitors. Make sure you get C23 the right way round. Solder ICs 1, 2 and 4 in place next, ensuring they are correctly oriented. Install a socket for IC3 next, but don't insert the EPROM yet. Now solder in the three diodes, followed by the relay. Check that the diodes are inserted the right way round. Now solder Q1 in place, then the 44-pin right-angle socket. Last of all, plug in the EPROM.

Put the decoder board aside and tackle the modem board next. As before, start by soldering in the links. There are only two (contrary to what you can see in the pictures — a prototype, later modified). One is ▶

2 of 7.



HOW IT WORKS — ETI-756

There are two sections to the project, each contained on separate boards: the 'decoder' (or decoder/control) board and the 'modem' board. They are powered from the +9 V and +5 V supply rails of the VZ200. Let's take each section separately.

DECODER BOARD

This decodes five ports and contains the software in EPROM plus the transmitter control relay. IC1 decodes address lines A11-A13, five of its Q outputs selecting the EPROM, transmit control and receive control circuitry as required. The outputs are 'enabled' when 1-1-0 appears on A14, A15 and the MREQ line.

Serial baudot data for transmit and receive goes in and out on bit seven of the VZ200's data buss (D7).

When you select transmit operation from the VZ200, the relay closes the push-to-talk (PTT) contacts, turning on your transmitter. When you send text, the data is sent via D7 and to the modulator board via the flip-flop IC2b and the $\overline{\text{TXD}}$ line.

When you select receive operation, the pulses from the demodulator on the modern board come in via the $\overline{\text{RXD}}$ line, and are gated onto D7 via IC4d and c. Note that, on selecting receive operation, Q1 gets turned off and the relay PTT contacts open, turning off your transmitter.

Diodes D4 and D5 make a simple OR gate, allowing the 'chip enable' pin of the EPROM to be activated when either the lower or upper 1K block of the EPROM is selected.

IC2 is a flip-flop that sets up the transmit control. Its outputs must be preset on power-up, hence the two 'clear' pins (CD1 and CD2) are initially clamped to 0 V on power-up because C23 is initially uncharged. It will charge via R3, by which time the Q outputs of IC2 will be correctly set.

MODEM BOARD

The receiver portion comprises two op-amps from IC9 (a and b), and IC6, an XR2211 PLL chip.

The two op-amps are set up as bandpass filters, each with the centre frequency offset so that their adjacent skirts just overlap. The filter Qs were chosen to provide good skirt selectively so that noise and interference in the received channel do not adversely affect the demodulator's operation. The lower roll-off is at about 2070 Hz, the upper roll-off at about 2350 Hz, neatly encompassing the standard mark and space tones used in amateur RTTY of 2125 and 2295 Hz. Note that 1% components are used for the critical filter components.

The filter output, from pin 7 of IC9, couples to the PLL input via C11. The PLL centre frequency is determined by C10 (chosen for its low temperature coefficient — see main text) and R14/RV2. The latter sets the PLL on frequency.

The PLL's dc 'error' signal toggles from high to low as the incoming audio switches from 2295 Hz to 2125 Hz. This output is the $\overline{\text{RXD}}$ line, sending the baudot bit stream to the VZ200 via the decoder board.

The XR2211 provides a 'lock detect' pin and this is used to drive a LED indicator via a transistor buffer (Q2).

The audio input to the demodulator is taken from the receiver's speaker. The level is first attenuated and then clipped with back-to-back diodes, D2 and D3. The 500 mV pk-pk level here is further attenuated (via R34/R35) before being applied to the input of the filter stages.

The modulator comprises a 'Walsh Function' generator, which digitally generates a pseudo-sinewave, followed by a buffer filter. The Walsh Function generator consists of IC5, a 555 timer running at ten times the required output frequency, followed by a 4017 decade counter. The 555 is toggled between the two required frequencies (21 250 Hz and 22 950 Hz) by switching extra resistance across the 555's timing resistor, thus raising its frequency of oscillation. This is done using a 4066 CMOS switch to switch RV3-R53 in parallel with RV1-R9. The $\overline{\text{TXD}}$ line toggles the 4066.

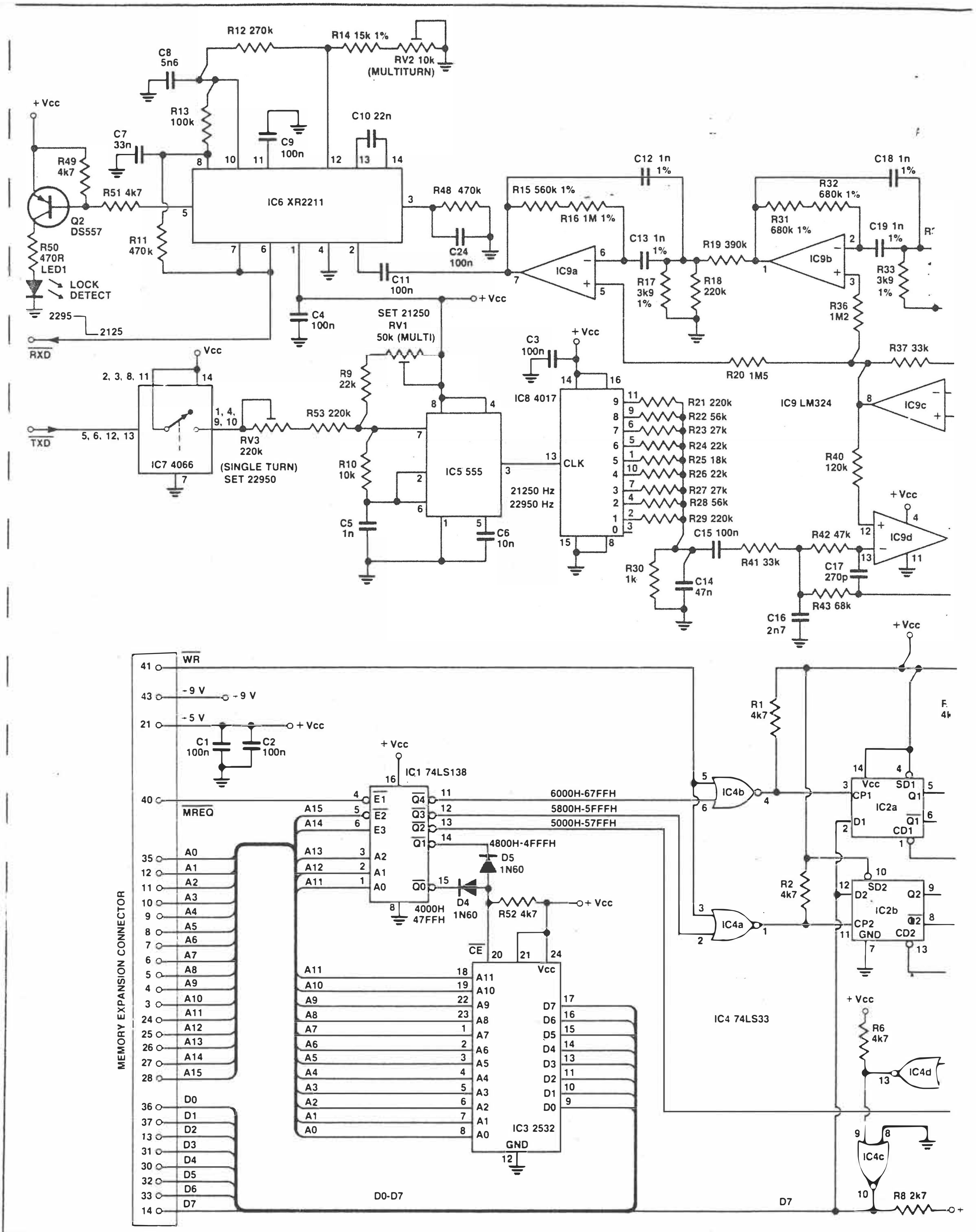
The output of the 555 drives the clock input of the 4017. The decade counter's outputs are all 'chained' via resistors R21-R29 so that the voltage across R30 'steps' up and down, depending on the ratio of high-to-low outputs of the 4017. The CR network of C14-R30 provides some high frequency roll-off.

One op-amp from IC9 (d) provides a buffer/filter, 'rounding off' the digitally generated sinewave before it is passed to the transmitter's mic input. C15 provides ac coupling to the op-amp input. C17 prevents RF from creating havoc in the mic line.

The op-amps require a half-supply rail for their non-inverting inputs and this is provided by IC9c and the divider R38-R39. C21 bypasses the half-supply divider.

Trimpot RV1 sets the low tone, while RV3 sets the high tone of the modulator. Note that RV3 is only a single-turn trimpot, while RV1 is a multi-turn type.

4 of 7.



ETI Nov 84 p 108.

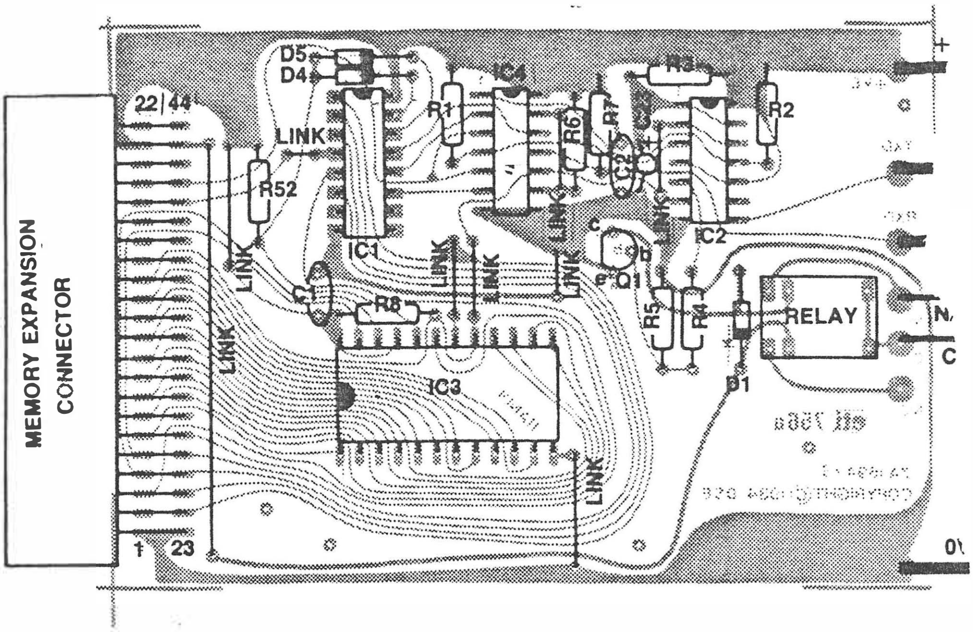
3 of 7.

located between R9 and R10, the other between R17 and R46. Use 22g tinned copper wire. Insert all the resistors next. Follow with the two diodes, Q2 and LED1 — making sure you get them all the right way round. Now solder all the ICs in place, seeing that you have them correctly oriented before soldering. With IC6, IC7 and IC8, solder the ground pins first, followed by the Vcc pin, and then all the remaining pins. This prevents any static or leakage current failure problems with the CMOS during construction.

The trimpots can be soldered in place next. Note that RV3 (SET 22 950) is a signal turn, vertical-mounting type, not a 10-turn trimpot like the others (and as seen in the pictures).

All the capacitors are soldered in place last. See that the two tantalums (C22 and C23) are correctly oriented.

Before proceeding further, *give each board a thorough check*. See that all the semiconductors and other polarised components are around the right way and that there are no solder bridges between closely-spaced pads — particularly around the IC pins. Remedy any problems.



The following is a summary of the commands for this system:

TRANSMIT COMMANDS

When called, the following commands are inserted into the type — ahead buffer ready for transmission.

- SHIFT Q — Transmit buffer #1.
- SHIFT W — Transmit buffer #2.
- SHIFT E — Transmit buffer #3.
- SHIFT R — Transmit buffer #4.
- SHIFT T — Transmit buffer #5.
- SHIFT 0 — Transmit buffer #0 (WRU buffer).
- SHIFT A — Transmit a row of RYs (32 characters).
- SHIFT I — Transmit "STATION IDENTIFICATION" along with your callsign.
- SHIFT P — Transmit "PLEASE KK KK KK" to terminate a call.
- SHIFT D — Transmit "DE" along with your callsign.
- SHIFT F — Transmit "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789".
- SHIFT C — Transmit a row of CQs (32 characters) along with your callsign.
- SHIFT 0 — Transmit your callsign only.
- SHIFT 3 — Terminate the transmission at this point and exit to receive mode. (SHIFT 3 produces a #).

IMMEDIATE COMMANDS

These commands operate in both transmit and receive modes.

- SHIFT Z — Toggle from the current mode to the alternative mode; i.e.: from TX to RX or from RX to TX.
- SHIFT — Exit from the current mode to the menu.
- SHIFT U — Enable/disable the WRU mode. The current status is displayed on the command line at the top of the screen.
- SHIFT H — Enable/disable the PRINTER mode. The current status is displayed on the command line at the top of the screen.
- SHIFT M — Backspace key. Deletes the last character typed.
- SHIFT S — Change the BAUD RATE.
- SHIFT B — Clears the internal printer buffer.
- SHIFT G — Exits the current mode and restarts at the callsign entry mode.
- SHIFT (RET) — Inserts a CR/LF into the internal printer buffer, forcing it to dump its contents to the printer.

If all's well, link the two boards with short lengths of hookup wire, as per the wiring diagram, and wire them to the DIN socket. Colour-coding the wires helps identify them, now as well as later when you may need to fault-find on the unit. Bolt the plastic spacers to the decoder board and screw the two boards together 'back-to-back'. If you're satisfied all is well, screw the assembly into the case bottom via the holes provided on the decoder board. This board faces down (components face the case). Leave the lid hanging loose so that the trimpots may be adjusted.

Aligning the unit

We will align the transmitter first, as the transmitter will be used to align the receiver.

Transmit alignment.

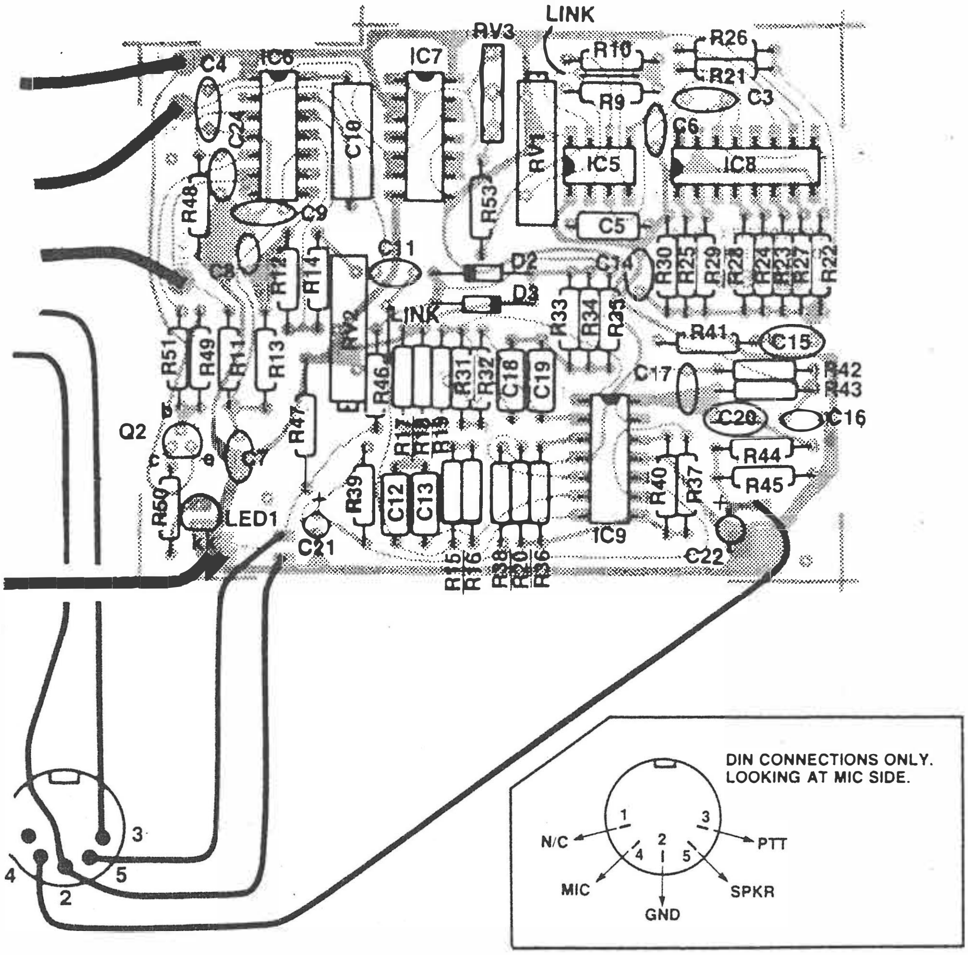
1) Cut the link connecting the two pads marked $\overline{TXD}$ on both boards. Solder a 10 cm length of wire to the modem board $\overline{TXD}$ pad.

2) Connect a frequency counter to pin 3 of IC5 (555).

3) Link the 10 cm wire to ground, and adjust RV1 for a frequency of 21 250 Hz.

4) Now link the wire to +5 V, and adjust RV3 for a frequency of 22 950 Hz.

5) Repeat steps 3 and 4 several times as necessary to ensure frequencies remain accurate when the wire is toggled between ground and +5 V.

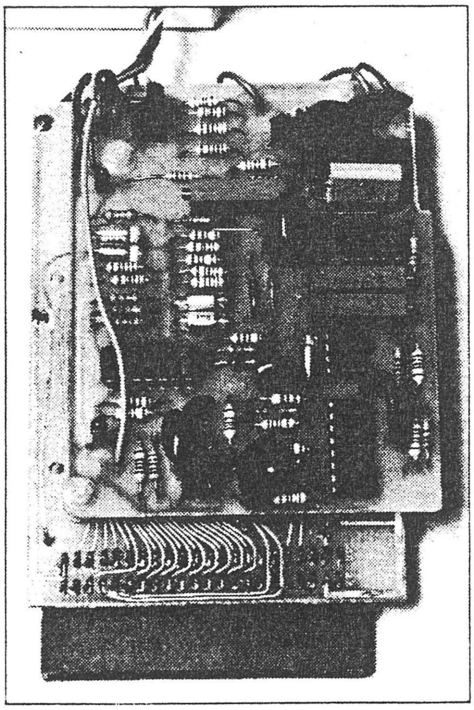


PC BOARD
The printed circuit artwork was done by Dick Smith Electronics and copyright is held by them. Hence, we have not reproduced the board pattern. Complete kits are available from Dick Smith stores.

Receiver alignment.
1) Wire a link connecting TX audio output to RX audio input.

2) Connect an audio generator to the wire used in the transmitter alignment.

3) Set the generator for a square wave, 0 dB attenuation, maximum amplitude, and a frequency of about 22 Hz. (This simulates a speed of approximately 45 baud).



Modem board. The receiver demodulator and transmitter modulator are contained on this board, mounted on the rear of the decoder board. Note the indicator LED.

PARTS LIST — ETI-756

Resistors ....all ¼W, 5% unless noted
R1-6,49,51,52 ....4k7
R7, R8 ....2k7
R9,24,26...............22k
R10........................10k
R11, R48 ....470k
R12........................270k
R13........................100k
R14........................15k, 1%
R15........................560k, 1%
R16........................1M, 1%
R17, R33 ....3k9, 1%
R18,21,29,53.........220k
R19........................390k
R20........................1M5
R22, R28 ....56k
R23,27,34..............27k
R25........................18k
R30,R46 ....1k
R31, R32 ....680k, 1%
R35........................330k
R36........................1M2
R37, R41 ....33k
R38, 39, 43...........68k
R40........................120k
R42........................47k
R44........................3k3
R45........................560R
R47........................33R
R50........................470R
RV1 ....50k multiturn trimpot
RV2 ....10k multiturn trimpot
RV3 ....200k vert. mount trimpot

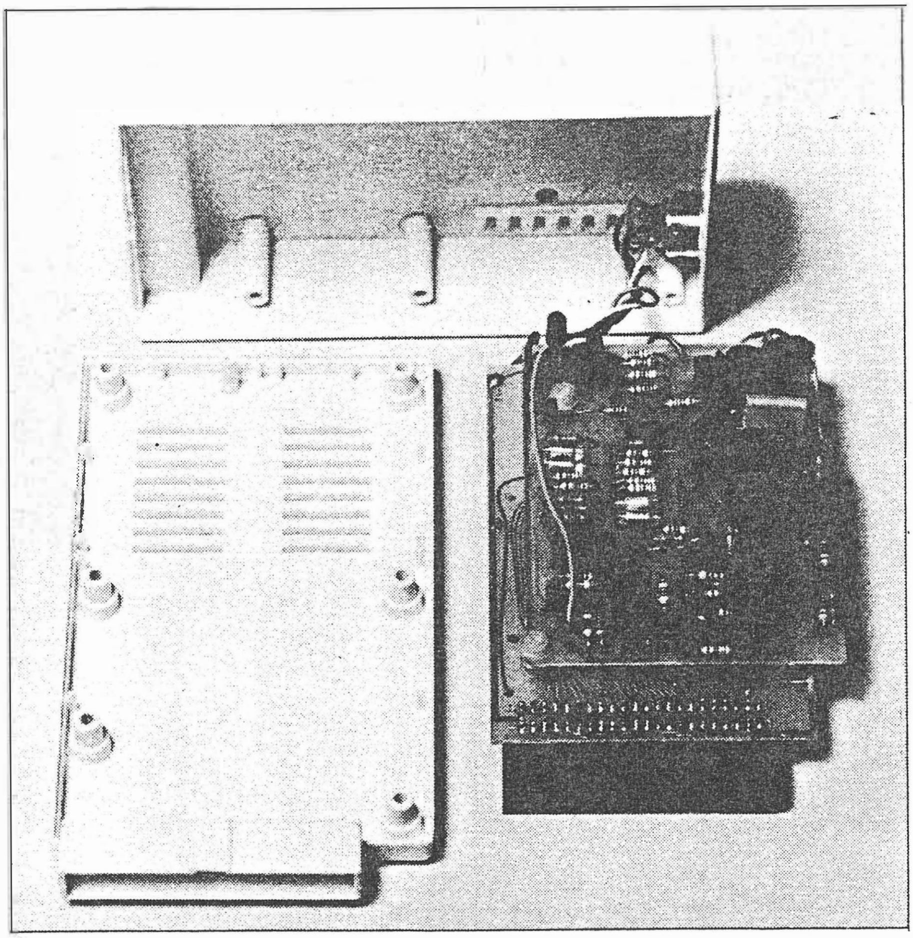
Capacitors
C1-4,9,24 ....100n ceramic
C5,12,13,18,19......1n, 1% styro
C6..........................10n ceramic
C7..........................33n greencap
C8..........................5n6 greencap
C10........................22n/400 V metallised poly cap. (mpc)
C11,15,20.............100n greencap
C14........................47n greencap
C16........................2n7 greencap
C17........................270p, 1% styro
C21........................470n electro (pc mount)
C22........................330n/10 V tant.
C23........................470n/10 V tant.

Semiconductors
D1,2,3....................1N914, 1N4148
D4, D5 ....1N60
LED1 ....5 mm red LED
Q1 ....DS548
Q2 ....DS557
IC1.........................74LS138
IC2.........................74LS74
IC3.........................2532 EPROM, "VZRTTY"
IC4.........................74LS33
IC5.........................DS555
IC6.........................XR2211
IC7.........................4066
IC8.........................4017
IC9.........................LM324, μA324

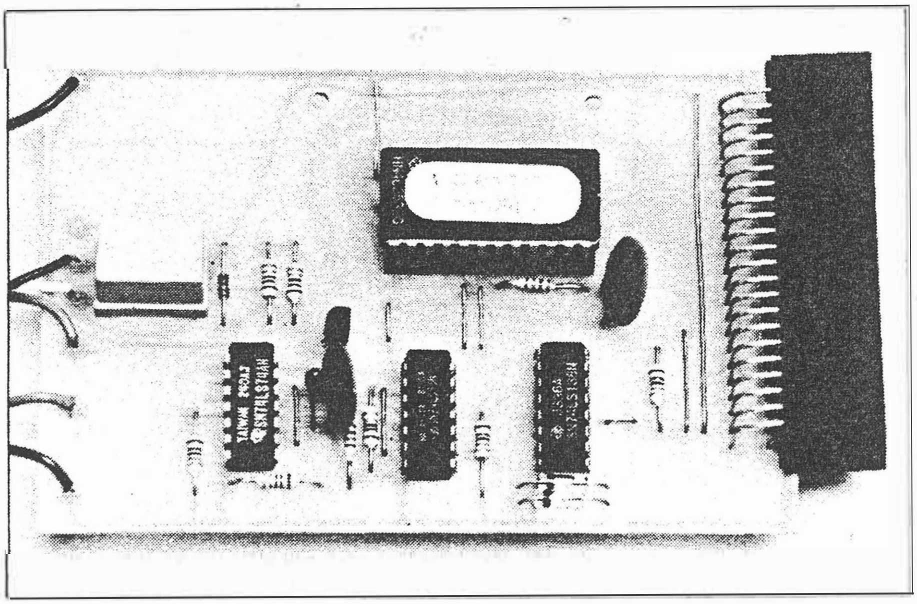
Miscellaneous
ETI-756 a and b pc boards (D.S.E. ZA1694 and ZA1695); 44-way edge connector (D.S.E. ZA 4107); case — Vitec RAM PAK case (D.S.E. ZA4663); Relay — mini 12 V DPDT type (D.S.E. S 7112); 5-pin DIN socket (D.S.E. P1552); three plastic spacers; nuts, bolts, hookup wire, etc.

Price estimate: $70-$75

6 of 7.



Insides out. The two boards mount inside a case from the VZ200's manufacturer. The bottom of the case is shown at left. The decoder board mounts to this, the modem board being mounted to the decoder board. Note the hole for the indicator in the case top.



Decoder board. There's not much to it. This unit interfaces the project to the VZ200 and contains the software in EPROM.

4) Connect a CRO to pin 7 of IC6 (XR2211).

5) Adjust RV2 for a squarewave of equal mark/space ratio.

6) Set the generator for a frequency of about 50 Hz. Check that the signal on pin 7 of IC6 is still a squarewave of equal mark/space ratio. If not, readjust RV2, then check again on 22 Hz.

7) Disconnect the generator.

8) Link the wire to ground. Pin 7 of IC6 should go logic high.

9) Link the wire to +5 V. Pin 7 of IC6 should go logic low.

That covers the alignment details. All that remains is to reconnect the two pads labelled $\overline{\text{TXD}}$ and disconnect the link connecting the audio input to audio output.

Final testing

After powering up, go to receive mode. Using SHIFT Z, toggle between receive and transmit modes. You should hear the transmit/receive relay open and close. The relay should be in the open condition on receive.

While in the transmit mode, the idle tone should be 2125 Hz, and the $\overline{\text{TXD}}$ pad should be a logic high. When typing, $\overline{\text{TXD}}$ should show low-going data, and the tone should toggle to 2295 Hz in sync. This tone will probably be too low in level to be read by a counter at the audio output pin, but it can be read on pin 3 of IC5 (555). (NOTE: This reading is 10 times the final frequency, so don't be fooled.)

Try out

Plug the project into the VZ200 expansion slot with the decoder board components facing *down*. Failure to observe this could result in the unit being damaged.

Once the module is fitted, turn your VZ200 on. If your VZ200 has Version 2.1 BASIC, you should hold down the CTRL key as you turn on, or else the display will contain inverse characters. If all is well, the VZ-200 should display

★ VZ-200 RTTY ★
★ TERMINAL PACK ★

followed by a copyright message. If not, power down immediately, and check the project for errors.

If all is well, you are ready to align the receive and transmit sections.

Before starting the alignment procedure, however, run through the general operation to ensure the software decoding is working fully.

PART 2: In the next instalment, we cover the overall operation of the unit, plus a listing of the software and a guide to its workings. ●

 7 of 7.

A 'GLASS TELETYPE' USING THE VZ200

Part 2

Keeping up with the popularity of radioteletype transmission has prompted a few projects from us. Last month we published Part 1 of project 756, designed and developed by Dick Smith's R & D Department to add on an RTTY to the accessible VZ200. This article completes that project and should get you on the airwaves.

IN THE FIRST PART of this article we described the construction of the hardware for your VZ200 RTTY interface. Hopefully by now you have a working RTTY interface plugged into your computer and are rarin' to get on the airwaves and start decoding these dots and dashes. In this part we give the final hookup information and details on using the software as well as a full software listing. Start warming up those transceivers and read on . . .

Now comes the time to connect your transceiver to the interface. Connection is made through the five-pin DIN plug on the rear panel. Wire the TX output and PTT pins to a microphone plug, and the RX input to a speaker plug. You will probably prefer to fit an extension speaker so you can monitor the received signals. Plug the microphone and speaker plugs into your transceiver anbd adjust the receive volume for a comfortable listening level to start with. High receive volume with the mute open on FM, will cause random characters to appear on the screen. This is to be expected if you over-drive the preamp/filters. These high volume levels are not required, and normal operation will require the volume to be no more than normal listening level.

If operating on VHF/UHF, the RTTY signals will probably be FM. This makes things easy, as the received tones will be of the correct frequency. Simply select the channel and adjust the volume. The 'lock detect' LED will light when a signal is being received correctly.

When operating on HF using SSB, care is required in tuning to the correct frequency. The LED will indicate when you are close. If you can't resolve it, try the other side-band.

This RTTY interface is designed to use a shift of 170 Hz. If you wish to receive commercial TTY (many of which use larger shifts), simply tune into one tone only. The 'lock' effect of the XR2211 will ensure correct data reception. Again, if you have difficulty, try the other sideband, the other tone, or another baud rate. NOTE: When receiving commercial, wide-shift TTY, the LED will flash in time with the data, due to the out-of-lock condition on one tone.

The normal specifications for Amateur RTTY are as follows

Mark (logic low) .... 2125 Hz
Space (logic high) .... 2295 Hz
Shift .... 170 Hz
Speed .... 45.45 baud
Idle: logic high
1 start bit
5 data bits
1.5 stop bits

That concludes the general operation of the RTTY interface. Those Sydney operators who are new to RTTY will find plenty of activity on the Sydney RTTY repeater on 146.675 MHz. There is also a RTTY simplex channel on 146.600 MHz You will find many operators only too glad to encourage newcomers to this mode of communications.

GENERAL OPERATION

Entering your callsign.
On power-up, your VZ200 RTTY interface will introduce itself. To continue, press any key. You will then be asked to enter your callsign. You may enter anything up to 64 characters but it is recommended that if you wish to use the WRU mode, you use the following format:

enter your callsign
VK2FGH (PETER)

There should be no leading space before the callsign and there should be at least one space after the callsign. Apart from that, you may add anything you like up to 64 characters total. This enables your callsign to be used as the WRU code. You may wish to use another code instead. If so, it must not be longer than a normal callsign (i.e: six letters) although it may be shorter, and it must always be followed by a space character. If you press <RETURN> at this point instead of entering text, the callsign buffer will contain a null and any attempt to send a callsign will give no response. The disadvantage of this is that your WRU system (when activated), instead of being selective, will respond to any WRU sent.

Loading the programmable buffers.
Once you have entered your callsign, press <RETURN> and you will enter the buffer entry mode. In this mode, you are able to enter text into any of the six programmable buffers. Each buffer may contain up to 64 characters. You may start entering text by typing the number of the buffer you require. Your VZ200 will display the buffer number you have selected. Simply enter your text as you require.

Note: the SHIFT M command is used for the backspace key.

Press <RETURN> when you are finished, and your buffer is programmed. Repeat the process for each buffer you require to program, including the WRU buffer (buffer 0). When you have finished, press SHIFT X to enter the MENU.

Menu mode.
From the MENU you are able to enter the three main operation modes, i.e: receive mode, transmit mode, and buffer entry mode. You can return to the menu at any time from any of these modes by using SHIFT X.

Receive mode.
In this mode you are able to receive RTTY. The first thing you will notice is the command line at the top of the screen. This line tells you the current status of the system. In the RECEIVE mode it will display RECEIVE MODE on the left. On the right will be the number 45. This is the current BAUD rate. The system will always default to 45.45 baud.

The command line is also used to display the current status of the PRINTER and WRU modes. These modes always default to the OFF status.

To demonstrate this, hold down the ►

1 of 5.

PROGRAM LISTING

```
ADDR   0  1  2  3   4  5  6  7   8  9  A  B   C  D  E  F

4000:  AA 55 E7 18  21 FF 7F F9  F3 3E 0D 21  09 80 77 11
4010:  0A 80 01 80  01 ED B0 3E  EC 32 08 80  AF 32 06 80
4020:  32 07 80 CD  EF 49 CD D9  49 CD C9 01  21 F7 43 CD
4030:  A7 28 CD 20  47 FE 00 28  F9 CD C9 01  21 AC 44 CD
4040:  A7 28 21 8F  81 C3 2E 46  CD DA 41 CD  C9 01 AF 32
4050:  00 60 21 F6  81 22 F4 81  77 11 F7 81  01 01 04 ED
4060:  B0 CD CA 46  32 F2 81 21  60 42 CD A3  45 21 90 42
4070:  CD A7 28 21  D6 42 CD A7  28 21 09 43  CD A7 28 CD
4080:  F4 2E FE 31  CA 93 40 FE  32 CA 1F 48  FE 33 CA B5
4090:  45 18 EC AF  32 00 60 CD  C9 01 21 1E  42 CD A3 45
40A0:  CD 77 46 CD  D9 49 CD EF  49 21 C0 71  22 00 80 21
40B0:  E0 70 22 F0  81 CD 70 4B  AF 32 05 80  CD 23 49 3A
40C0:  05 80 FE FF  CA 28 48 CD  8F 46 1E 00  3A 00 50 CB
40D0:  7F 28 E9 CD  23 49 3A 05  80 FE FF CA  28 48 CD 8F
40E0:  46 3A 00 50  CB 7F 20 EB  0E 08 CD 1E  45 CD 18 45
40F0:  3A 00 50 CB  17 CB 11 38  05 CD 18 45  18 F2 CD 18
4100:  45 21 E0 46  06 00 16 00  79 FE 1B 20  02 1E 01 FE
4110:  1F 20 02 1E  00 CB 21 19  09 7E FE 09  FA 2E 41 FE
4120:  0D 20 02 1E  00 CD 68 45  CD 30 41 CD  5B 41 18 A3
4130:  E5 D5 F5 3A  07 80 FE 00  28 1D 37 3F  ED 5B F8 85
4140:  21 38 86 ED  52 28 10 F1  FE 0C 38 08  2A F8 85 77
4150:  23 22 F8 85  D1 E1 C9 F1  D1 E1 C9 F5  D5 F5 3A 06
4160:  80 FE 00 20  03 F1 18 2F  21 D0 81 11  DA 81 01 0A
4170:  00 ED B0 21  DA 81 11 D1  81 01 09 00  ED B0 11 D0
4180:  81 F1 12 21  E4 81 11 D9  81 1A BE 20  0A 23 1B 7E
4190:  FE AA 28 06  1A 18 F3 D1  F1 C9 D1 F1  21 0C 42 ED
41A0:  5B F4 81 01  06 00 ED B0  ED 53 F4 81  3E 06 32 F2
41B0:  81 CD F6 41  3E 0D CD 30  41 3A 00 50  CB 7F 20 F9
41C0:  0E 32 CD 1E  45 CD 18 45  3A 00 50 CB  7F 20 EA 0D
41D0:  20 F3 3A 05  80 2F 32 05  80 C9 21 8F  81 11 E4 81
41E0:  7E 12 FE 20  23 13 28 02  18 F6 06 04  21 FC 44 7E
41F0:  12 23 13 10  FA C9 06 0C  21 7F 00 C5  01 64 00 CD
4200:  5C 34 01 00  20 CD 60 00  C1 10 ED C9  95 0D 0D 90
4210:  0D 23 57 52  55 00 20 20  20 00 50 52  54 00 52 45
4220:  43 45 49 56  45 20 4D 4F  44 45 20 20  20 20 20 20
4230:  20 20 20 20  20 20 20 20  20 20 20 20  20 20 00 54
4240:  52 41 4E 53  4D 49 54 20  4D 4F 44 45  20 20 20 20
4250:  20 20 20 20  20 20 20 20  20 20 20 20  20 20 20 00
4260:  20 20 20 20  20 20 20 20  20 56 5A 2D  32 30 30 20
4270:  20 52 54 54  59 20 20 20  20 20 20 20  20 20 20 00
4280:  34 35 20 00  35 30 20 00  37 35 20 00  31 31 30 00
4290:  0D 0D 0D 20  20 20 20 20  20 20 20 20  20 20 20 20
42A0:  4D 45 4E 55  0D 0D 20 20  20 31 29 20  52 45 43 45
42B0:  49 56 45 20  20 20 32 29  20 54 52 41  4E 53 4D 49
42C0:  54 0D 20 20  20 33 29 20  4C 4F 41 44  20 42 55 46
42D0:  46 45 52 53  0D 00 0D 20  20 50 52 45  53 53 20 D3
42E0:  C8 C9 C6 D4  20 D8 20 41  54 20 41 4E  59 20 54 49
42F0:  4D 45 0D 20  20 20 20 20  20 20 20 20  20 20 46 4F
4300:  52 20 4D 45  4E 55 0D 0D  00 0D 20 20  20 20 20 20
4310:  20 20 20 53  45 4C 45 43  54 20 28 31  2D 33 29 0D
4320:  00 0D 0D 20  20 20 20 20  20 42 55 46  46 45 52 20
4330:  49 4E 50 55  54 20 52 4F  55 54 49 4E  45 0D 0D 0D
4340:  0D 0D 20 20  45 4E 54 45  52 20 42 55  46 46 45 52
4350:  20 4E 55 4D  42 45 52 20  28 30 2D 35  29 00 20 57
4360:  52 55 20 42  55 46 46 45  52 0D 00 20  42 55 46 46
4370:  45 52 20 23  31 0D 00 20  42 55 46 46  45 52 20 23
4380:  32 0D 00 20  42 55 46 46  45 52 20 23  33 0D 00 20
4390:  42 55 46 46  45 52 20 23  34 0D 00 20  42 55 46 46
43A0:  45 52 20 23  35 0D 00 44  45 20 0D 43  51 20 43 51
43B0:  20 43 51 20  43 51 20 43  51 20 43 51  20 43 51 20
43C0:  43 51 20 43  51 20 43 51  20 0D 20 50  4C 53 20 4B
43D0:  4B 20 4B 4B  20 4B 4B 0D  52 59 52 59  52 59 52 59
43E0:  52 59 52 59  52 59 52 59  52 59 52 59  52 59 52 59
43F0:  52 59 52 59  52 59 0D 0D  0D 20 20 20  20 20 20 20
4400:  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 2A
4410:  2A 2A 0D 20  20 20 20 20  20 20 2A 20  20 56 5A 2D
4420:  32 30 30 20  20 52 54 54  59 20 20 2A  0D 20 20 20
4430:  20 20 20 20  2A 20 54 45  52 4D 49 4E  41 4C 20 20
4440:  50 41 43 4B  20 2A 0D 20  20 20 20 20  20 20 2A 2A
4450:  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 2A
4460:  0D 0D 20 20  20 20 20 20  20 20 20 20  20 20 20 20
4470:  46 52 4F 4D  0D 20 20 20  20 20 44 49  43 4B 20 53
4480:  4D 49 54 48  20 45 4C 45  43 54 52 4F  4E 49 43 53
4490:  0D 20 20 20  20 20 20 20  43 4F 50 59  52 49 47 48
44A0:  54 20 28 43  29 20 31 39  38 34 0D 00  0D 45 4E 54
44B0:  45 52 20 59  4F 55 52 20  43 41 4C 4C  53 49 47 4E
44C0:  20 3A 0D 00  54 48 45 20  51 55 49 43  4B 20 42 52
44D0:  4F 57 4E 20  46 4F 58 20  4A 55 4D 50  53 20 4F 56
44E0:  45 52 20 54  48 45 20 4C  41 5A 59 20  44 4F 47 20
44F0:  30 31 32 33  34 35 36 37  38 39 20 0D  57 52 55 AA
4500:  53 54 41 54  49 4F 4E 20  49 44 45 4E  54 49 46 49
4510:  43 41 54 49  4F 4E 20 0D  CD 1E 45 CD  23 49 C5 3A
4520:  08 80 06 0B  10 FE 3D 20  F9 C1 C9 2A  00 80 FE 0D
4530:  28 05 FE 20  30 23 C9 ED  5B 20 78 D5  22 20 78 3E
4540:  0D 21 02 80  BE 28 06 CD  3A 03 32 02  80 2A 20 78
4550:  22 00 80 D1  ED 53 20 78  C9 FE 40 38  02 D6 40 77
4560:  32 02 80 23  22 00 80 C9  E5 D5 C5 F5  CD 2B 45 7C
4570:  FE 71 28 02  18 06 7D FE  E0 D4 81 45  F1 C1 D1 E1
4580:  C9 F5 21 40  71 11 20 71  01 A0 00 ED  B0 21 C0 71
4590:  3E 20 77 11  C1 71 01 20  00 ED B0 21  C0 71 22 00
45A0:  80 F1 C9 11  00 70 7E FE  00 C8 FE 40  30 02 C6 40
45B0:  12 23 13 18  F1 CD C9 01  21 21 43 CD  A7 28 21 D6
45C0:  42 CD A7 28  CD 63 4B CD  20 47 CD 79  4B FE 01 CA
45D0:  48 40 FE 30  28 16 FE 31  28 1D FE 32  28 24 FE 33
45E0:  28 2B FE 34  28 32 FE 35  28 39 18 DB  21 5E 43 CD
45F0:  A7 28 21 09  80 18 37 21  6B 43 CD A7  28 21 4A 80
4600:  18 2C 21 77  43 CD A7 28  21 8B 80 18  21 21 83 43
4610:  CD A7 28 21  CC 80 18 16  21 8F 43 CD  A7 28 21 0D
4620:  81 18 0B 21  9B 43 CD A7  28 21 4E 81  18 00 06 40
4630:  CD 70 4B E5  C5 CD 20 47  C1 FE 01 28  36 FE 00 28
4640:  F3 FE 08 E1  28 12 77 FE  0D CA B5 45  23 CD 3A 03
4650:  CD 70 4B 10  DE C3 B5 45  78 FE 40 28  D1 3E 08 CD
4660:  3A 03 3E 20  CD 3A 03 3E  08 CD 3A 03  04 2B CD 70
4670:  4B 18 C0 E1  C3 48 40 3A  08 80 FE EC  CA 65 4C FE
4680:  D7 CA 47 4C  FE 8D CA 51  4C FE 61 CA  5B 4C C9 F5
4690:  C5 E5 D5 CD  C4 05 CB 47  20 18 3A FA  85 FE 00 28
46A0:  11 CD DB 46  FE 0D 20 07  3E 0A 32 FA  85 18 E4 CD
46B0:  B7 46 D1 E1  C1 F1 C9 11  FA 85 21 FB  85 01 40 00
46C0:  ED B0 2A F8  85 2B 22 F8  85 C9 AF 21  FA 85 22 F8
46D0:  85 77 11 FB  85 01 40 00  ED B0 C9 D3  0E D3 0D C9
46E0:  05 05 54 35  0D 0D 4F 39  20 20 48 00  4E 2C 4D 2E
46F0:  00 00 4C 29  52 34 47 24  49 38 50 30  43 3A 56 3D
4700:  45 33 5A 2B  44 00 42 3F  53 27 59 36  46 25 58 2F
4710:  41 2D 57 32  4A 27 04 04  55 37 51 31  4B 28 07 07
4720:  21 FE 68 0E  08 06 06 7E  F6 04 1F 30  4F 10 FB CB
4730:  05 0D 20 F1  06 04 21 DF  68 7E CB 57  28 23 CB 05
4740:  7E CB 57 28  20 CB 05 7E  CB 57 28 1D  CB 05 CB 05
4750:  CB 05 7E CB  57 28 16 3E  FF 32 04 80  AF 32 03 80
4760:  C9 0E 03 18  17 0E 02 18  13 0E 01 18  0F 3A 03 80
4770:  CB CF 32 03  80 3E FF 32  04 80 AF C9  21 B0 47 1E
4780:  00 3A 03 80  CB 4F 28 02  1E 30 3E 08  91 4F 3E 06
4790:  90 47 CD A3  47 83 06 00  4F 09 7E 21  04 80 BE 28
47A0:  D9 77 C9 AF  B9 28 07 C6  06 0D 28 02  18 F9 80 C9
47B0:  54 57 20 45  51 52 47 53  20 44 41 46  42 58 20 43
47C0:  5A 56 35 32  20 33 31 34  4E 2E 20 2C  20 4D 36 39
47D0:  2D 38 30 37  59 4F 0D 49  50 55 48 4C  3A 4B 3B 4A
47E0:  75 72 00 73  71 74 85 87  00 84 89 8F  83 01 00 81
47F0:  0A 00 25 22  20 23 21 24  00 00 00 00  00 08 26 29
4800:  3D 28 90 27  00 82 91 95  88 94 86 3F  00 2F 2B 00
4810:  21 20 71 3E  20 77 11 21  71 01 C0 00  ED B0 C9 CD
4820:  C9 01 21 E0  70 22 F0 81  CD 10 48 3E  FF 32 05 80
4830:  AF 32 EF 81  21 3F 42 CD  A3 45 CD 77  46 CD D9 49
4840:  CD EF 49 21  C0 71 22 00  80 CD 70 4B  3E FF 32 00
4850:  60 06 96 CD  18 45 10 FB  CD 23 49 3A  05 80 FE 00
4860:  CC C6 48 CD  8F 46 CD 10  4A FE 00 28  EB FE 71 CA
4870:  CD 4B FE 72  CA D7 4B FE  73 CA E1 4B  FE 74 CA EB
4880:  4B FE 75 CA  F5 4B FE 81  CA 84 4B FE  82 CA 9B 4B
4890:  FE 84 CA 94  4B FE 88 CA  A5 4B FE 89  CA AF 4B FE
48A0:  8F CA B9 4B  FE 90 CA C3  4B FE 95 CA  8D 4B FE 23
48B0:  CA CE 48 CD  30 41 CD D4  48 CD 68 45  FE 0D CA 3E
48C0:  4B CD B7 4A  18 92 21 F6  81 7E FE 00  C0 F1 CD 63
48D0:  4B C3 93 40  F5 F5 FE 0D  28 1A FE 20  28 0E 3A EF
48E0:  81 FE 3E 28  0F 3C 32 EF  81 F1 F1 C9  3A EF 81 FE
48F0:  39 DA E5 48  AF 32 EF 81  F1 FE 0D 28  ED FE 20 28
4900:  08 3E 2D CD  68 45 CD B7  4A CD 5E 4B  CD 59 4B 3E
4910:  0D CD 68 45  F1 FE 20 28  08 F5 3E 01  32 EF 81 F1
4920:  C9 AF C9 D5  C5 CD 20 47  FE 00 28 66  FE 01 28 68
4930:  FE 0A CA B7  49 FE 87 CA  9E 49 FE 83  28 66 FE 86
4940:  CA CD 49 FE  94 CA C1 49  FE 85 28 5D  FE 91 28 5F
4950:  CD 3F 4A F5  E5 C5 FE 08  20 16 2A F2  81 7C B5 CA
4960:  03 4A 2B 22  F2 81 2A F4  81 2B AF 77  3E 08 18 1C
4970:  F5 ED 4B F2  81 21 FC 03  ED 42 7C B5  CA 0D 4A 2A
4980:  F2 81 23 22  F2 81 F1 2A  F4 81 77 23  22 F4 81 C1
4990:  E1 F1 C1 D1  CD 70 4A C9  C1 D1 F1 C3  48 40 CD 17
49A0:  4C C1 D1 C9  CD CA 46 18  F8 C1 D1 F1  C3 39 40 3E
49B0:  0D 2A F8 85  77 18 EA C1  D1 3A 05 80  2F 32 05 80
49C0:  C9 3A 06 80  2F 32 06 80  CD D9 49 18  D4 3A 07 80
49D0:  2F 32 07 80  CD EF 49 18  C8 3A 06 80  FE 00 20 05
49E0:  21 16 42 18  03 21 12 42  11 10 70 CD  A6 45 C9 3A
49F0:  07 80 FE 00  20 05 21 16  42 18 03 21  1A 42 11 15
4A00:  70 18 E8 C1  E1 F1 C1 D1  AF CD 70 4A  C9 F1 18 F3
4A10:  2A F2 81 7C  B5 C8 2B 22  F2 81 21 F6  81 7E FE 00
4A20:  C8 F5 21 F7  81 01 00 04  09 AF 77 21  F7 81 11 F6
4A30:  81 01 00 04  ED B0 2A F4  81 2B 22 F4  81 F1 C9 E5
4A40:  C5 D5 F5 2A  F0 81 7D FE  3F D2 6B 4A  21 60 70 11
4A50:  40 70 01 A0  00 ED B0 21  E0 70 3E 20  77 11 E1 70
4A60:  01 20 00 ED  B0 21 E0 70  22 F0 81 F1  D1 C1 E1 C9
4A70:  FE 08 28 25  FE 0D 28 05  FE 20 30 28  C9 ED 5B 20
```


2 of 5.

```
4A80:  78 D5 2A F0  81 22 20 78  3E 0D CD 3A  03 2A 20 78
4A90:  22 F0 81 D1  ED 53 20 78  C9 2A F0 81  3E 20 2B 77
4AA0:  22 F0 81 C9  FE 40 38 06  FE 60 30 02  D6 40 2A F0
4AB0:  81 77 23 22  F0 81 C9 D6  0A FE 03 CA  5E 4B D6 13
4AC0:  FE 40 D0 FE  03 CA D8 4A  FE 21 D2 D8  4A 08 FE FF
4AD0:  28 10 3E FF  08 C3 33 4B  08 FE 00 28  05 AF 08 C3
4AE0:  28 4B 08 CD  E7 4A C9 21  76 4C 01 00  00 4F 09 4E
4AF0:  06 06 CB 11  CB 11 CB 11  DA 07 4B C3  12 4B 10 F6
4B00:  CD 1C 4B CD  8F 46 C9 3E  FF 32 00 58  CD 18 45 C3
4B10:  FE 4A AF 32  00 58 CD 18  45 C3 FE 4A  3E FF 32 00
4B20:  58 CD 18 45  CD 1E 45 C9  F5 0E 1F CD  F0 4A F1 CD
4B30:  E7 4A C9 F5  0E 1B CD F0  4A F1 CD E7  4A C9 0E 02
4B40:  CD F0 4A C3  46 4B 0E 08  CD F0 4A C3  58 48 0E 04
4B50:  CD F0 4A 08  AF 08 C3 58  48 0E 08 C3  F0 4A 0E 02
4B60:  C3 F0 4A 01  FF BF CD 60  00 01 FF BF  CD 60 00 C9
4B70:  C5 01 FF 5F  CD 60 00 C1  C9 F5 C5 01  FF 2F CD 60
4B80:  00 C1 F1 C9  FD 21 AB 43  CD FF 4B 18  07 FD 21 00
4B90:  45 CD FF 4B  FD 21 A7 43  CD FF 4B FD  21 8F 81 CD
4BA0:  FF 4B C3 58  48 FD 21 CA  43 CD FF 4B  C3 58 48 FD
4BB0:  21 D8 43 CD  FF 4B C3 58  48 FD 21 C4  44 CD FF 4B
4BC0:  C3 58 48 FD  21 09 80 CD  FF 4B C3 58  48 FD 21 4A
4BD0:  80 CD FF 4B  C3 58 48 FD  21 8B 80 CD  FF 4B C3 58
4BE0:  48 FD 21 CC  80 CD FF 4B  C3 58 48 FD  21 0D 81 CD
4BF0:  FF 4B C3 58  48 FD 21 4E  81 CD FF 4B  C3 58 48 FD
4C00:  7E 00 FE 0D  C8 F5 CD 68  45 CD 30 41  CD 8F 46 CD
4C10:  B7 4A F1 FD  23 18 E8 3A  08 80 FE EC  28 0D FE D7
4C20:  28 10 FE 8D  28 13 FE 61  28 16 C9 3E  D7 32 08 80
4C30:  18 15 3E 8D  32 08 80 18  18 3E 61 32  08 80 18 1B
4C40:  3E EC 32 08  80 18 1E 11  1B 70 21 84  42 CD A6 45
4C50:  C9 11 1B 70  21 88 42 CD  A6 45 C9 11  1B 70 21 8C
4C60:  42 CD A6 45  C9 11 1B 70  21 80 42 CD  A6 45 C9 01
4C70:  FF FF CD 60  00 C9 08 02  00 04 00 00  00 0B 16 00
4C80:  1A 1E 09 00  11 06 18 07  17 0D 1D 19  10 0A 01 15
4C90:  1C 0C 03 0E  00 00 0F 00  13 00 18 13  0E 12 10 16
4CA0:  0B 05 0C 1A  1E 09 07 06  03 0D 1D 0A  14 01 1C 0F
4CB0:  19 17 15 11  FF 04 FF 04  FF 4C FF 4C  FF 4C FF 4C
4CC0:  FB 48 FB 48  FB 48 FB 08  FB 08 FB 00  FB 00 FB 00
4CD0:  FB 48 FB 08  FB 08 FB 08  FB 00 FB 00  FB 00 FB 00
4CE0:  FB 48 FB 48  FB 48 FB 08  FB 00 FB 00  FB 00 FB 00
4CF0:  FB 48 FB 08  FB 08 FB 08  FB 00 FB 00  FB 00 FB 00
4D00:  37 00 FF 04  FF 04 FF 04  FF 0C FF 0C  FF 0C FF 0C
4D10:  FF 04 FF 04  FF 04 FF 04  FF 0C FF 0C  FF 0C FF 0C
4D20:  FF 04 FF 04  FF 04 FF 04  FF 0C FF 0C  FF 0C FF 0C
4D30:  FF 04 FF 04  FF 04 FF 04  FF 0C FF 0C  FF 0C FF 0C
4D40:  FB 08 FB 08  FB 08 FB 08  FB 00 FB 00  FB 00 FB 00
4D50:  FB 08 FB 08  FB 08 FB 08  FB 00 FB 00  FB 00 FB 00
4D60:  FB 08 FB 08  FB 08 FB 08  FB 00 FB 00  FB 00 FB 00
4D70:  FB 08 FB 08  FB 08 FB 00  FB 00 FB 00  FB 00 FB 00
4D80:  37 0C FF 0C  FF 0C FF 0C  FF 4C FF 0C  FF 4C FF 0C
4D90:  FF 0C FF 04  FF 04 FF 04  FF 4C FF 4C  FF 4C FF 0C
4DA0:  FF 0C FF 0C  FF 0C FF 0C  FF 4C FF 0C  FF 4C FF 0C
4DB0:  FF 0C FF 04  FF 0C FF 04  FF 4C FF 0C  FF 4C FF 4C
4DC0:  FB 48 FB 08  FB 48 FB 08  FB 08 FB 08  FB 08 FB 08
4DD0:  FB 08 FB 08  FB 08 FB 08  FB 08 FB 08  FB 08 FB 08
4DE0:  FB 48 FB 08  FB 48 FB 08  FB 08 FB 08  FB 08 FB 08
4DF0:  FB 08 FB 08  FB 08 FB 08  FB 08 FB 08  FB 08 FB 08
4E00:  37 0C FF 0C  FF 0C FF 04  FF 0C FF 0C  FF 04 FF 04
4E10:  FF 04 FF 04  FF 04 FF 04  FF 0C FF 0C  FF 0C FF 04
4E20:  FF 0C FF 04  FF 0C FF 0C  FF 0C FF 04  FF 0C FF 04
4E30:  FF 04 FF 04  FF 04 FF 04  FF 0C FF 04  FF 0C FF 0C
4E40:  FB 08 FB 08  FB 08 FB 00  FB 08 FB 08  FB 08 FB 08
4E50:  FB 08 FB 00  FB 08 FB 00  FB 08 FB 08  FB 08 FB 08
4E60:  FB 08 FB 08  FB 08 FB 00  FB 08 FB 08  FB 08 FB 08
4E70:  FB 08 FB 08  FB 00 FB 00  FB 08 FB 08  FB 08 FB 08
4E80:  17 0C FF 0C  FF 04 FF 04  FF 0C FF 04  FF 04 FF 04
4E90:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4EA0:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4EB0:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4EC0:  FB 08 FB 00  FB 00 FB 00  FB 08 FB 08  FB 08 FB 08
4ED0:  FB 08 FB 00  FB 00 FB 00  FB 08 FB 08  FB 08 FB 00
4EE0:  FB 00 FB 00  FB 00 FB 00  FB 08 FB 08  FB 08 FB 08
4EF0:  FB 00 FB 00  FB 00 FB 00  FB 08 FB 00  FB 00 FB 08
4F00:  37 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4F10:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4F20:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4F30:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4F40:  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00
4F50:  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00
4F60:  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00
4F70:  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00  FB 00 FB 00
4F80:  37 0C FF 0C  FF 0C FF 04  FF 0C FF 04  FF 04 FF 04
4F90:  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4FA0:  FF 0C FF 04  FF 04 FF 04  FF 04 FF 04  FF 04 FF 04
4FB0:  52 4F 4D 20  43 4F 4E 54  45 4E 54 53  20 43 4F 50
4FC0:  59 52 49 47  48 54 20 28  43 29 20 31  39 38 34 20
4FD0:  44 49 43 4B  20 53 4D 49  54 48 20 45  4C 45 43 54
4FE0:  52 4F 4E 49  43 53 20 50  54 59 2E 20  4C 54 44 2E
4FF0:  20 41 55 53  54 52 41 4C  49 41 2E 00  00 00 00 00
```

NOTES & ERRATA

Nov '84, project 756, p 106: On page 107, last column, note that there are **nine** links on the decoder board, not eight. On the circuit diagram, page 109, C23 should read 470n; the Parts List is correct. On page 110, in the table under "Immediate Commands", the second command is SHIFT X. In the text on page 110, second last paragraph, the last sentence should read: "See that the two polarised capacitors (C21 and C22) are correctly oriented." Note that R7 is actually 2k7, as per the Parts List, not 4k7, as per the circuit.

THE SOFTWARE

There is an unused section in the VZ200 memory map between 4000H and 67FFH. This area was set aside for use with plug-in software packs. The RTTY unit fits into this area of memory.

For design simplicity, this section is decoded into five 2K blocks. The first two blocks are used for the main software routines. The other three blocks are used for receive data, transmit data and relay data.

All data transfer is done through bit 7 (D7). The software also uses a section of RAM starting at 8000H. This area is used to store volatile data such as buffers and flags.

Some useful RAM and EPROM addresses are given below.

RAM LOCATIONS

8000/01	Receive character cursor position
8005	Receive/transmit toggle flag
8006	WRU flag
8007	Printer flag
8008	Timing loop value (231 = 45.45 baud)
8009	Start of buffer 0
804A	Start of buffer 1
808B	Start of buffer 2
80CC	Start of buffer 3
810D	Start of buffer 4
814E	Start of buffer 5
818F	Start of callsign storage area
81F0/F1	Transmit cursor position
81F6	Start of keyboard input buffer

EPROM LOCATIONS

4000	EPROM entry point
4039	Callsign entry routine
45B5	Buffer entry routine
4048	Menu entry point
4093	Receive routine entry point
468F	Line printer routine entry point
4518	Delay routine
4923	Keyboard input and video processing routine
4810	Transmit entry point
484C	Transmit active point
4568	Transmit data video display routine
4AB7	ASCII to baud conversion
49B7	Toggle receive/transmit relay on/off
49C1	Toggle WRU on/off
49CD	Toggle printer on/off
499E	Change baud rate 45-50-75-110-45 etc.

OTHERS

5000	Receive data
5800	Transmit data
6000	Transmit/receive relay

3 of 5.

MODIFICATIONS TO VZ/RTTY DECODER TO IMPROVE PERFORMANCE ON WIDEBAND COMMERCIAL RTTY

The following changes to component values will allow less critical receiver tuning when decoding wideband commercial RTTY found on the HF bands.

While values are given for both 425 Hz and 850 Hz shifts, prototype units constructed for 850 Hz shift use were quite capable of resolving stations using 425 Hz shifts.

It should be noted that once these modifications have been performed, it is highly unlikely that the decoder will resovle 170 Hz shift amateur RTTY.

CHANGES FOR 850 Hz SHIFT (1450/2300 Hz)

i) Changes to filter stages

Change:

R35	from 300k	5%	to	180k	5%
R34	from 27k	5%	to	27k	1%
R33	from 3k9	1%	to	27k	1%
R32	from 680k	1%	to	1M	1%
R31	from 680k	1%	to	18k	5%
R19	from 390k	5%	to	100k	5%
R18	from 220k	5%	to	470k	5%
R17	from 3k9	1%	to	8k2	1%
R16	from 1M	1%	to	47k	5%

R15 no change.

ii) Changes to FSK decoder

Change:

RV2	from 10k		to	20k	
R14	from 18k	1%	to	15k	1%
R12	from 270k	5%	to	47k	5%
R11	from 470k	5%	to	1M5	5%

C7 from 330n to 39n

CHANGES FOR 425Hz SHIFT (1875/2300Hz)

i) Changes to filter stages

Change:

R35	from 330k	5%	to	220k	5%
R34	from 27k	5%	to	39k	1%
R33	from 3k9	1%	to	12k	1%
R32	from 680k	1%	to	820k	1%
R31	from 680k	1%	to	68k	1%
R19	from 390k	5%	to	150k	5%
R18	from 220k	5%	to	47k	5%
R17	from 3k9	1%	to	8k2	1%
R16	from 1M	1%	to	100k	1%

R15 no change

ii) Changes to FSK decoder

Change:

RV2	from 10k		to	20k	
R14	from 18k	1%	to	12k	1%
R12	from 270k	5%	to	100k	5%
R11	from 470k	5%	to	1M5	5%

C7 from 330n to 39n

SHIFT key and press U. The command line will display WRU. This indicates that the WRU mode is now active. Again press SHIFT U, and the WRU will no longer be displayed, indicating the WRU mode is disabled. Try the same with SHIFT H. This enables and disables the printer. Similarly, SHIFT 5 changes the BAUD rate.

The screen is split into two sections, each with independent scrolling. All received text is displayed on the bottom screen, while the top screen is used to display your typed text. You may type and receive simultaneously. The type ahead buffer can contain up to 1024 (1K) characters. Any data from the buffers may be added as you go by pressing the appropriate enable keys. A graphic block will be displayed as you type to show you that a buffer has been enabled. You may terminate your text with the '#' code. When this code is found during transmission, your system will automatically revert to the receive mode.

Transmit mode.
When the station you are communicating with has finished his transmission, you may reply to him by pressing

SHIFT Z

This sends your terminal to the transmit mode, enabling your transmitter, and sending the test you previously typed. You may continue typing if you wish. Your system will continue to send the stored text, including any programmed text, until it catches up with your typing, whereby it will follow the text as you type it. During all this time, the text is displayed on the bottom screen, along with the contents of any programmed buffers you may have enabled. Thus you can see everything being sent in its final form. You may exit to receive by using either

#
or
SHIFT Z

Note: SHIFT Z will not work if there is still data in the buffer waiting to be sent. This prevents you from accidentally terminating the transmission prematurely. If you wish to abort your transmission intentionally, use

SHIFT X

to get back to the menu.

WRU mode.
The WRU mode is a special feature included to add versatility to your system. To activate this mode, press

SHIFT U

The letters WRU will appear on the command line. When this mode is active, any station sending your callsign (or any other code entered on power-up), followed by the letters WRU, will activate your system. When this happens, your VZ200 will first Beep to let you know that your system is being called. After checking to ensure the frequency is clear, your transmitter will then activate automatically, sending 'STATION IDENTIFICATION DE < callsign>', along with any message stored in the WRU buffer (buffer # 0).

For example, if you had entered on power-up 'VK2FGH (PETER)' any station wishing to activate your WRU mode would need to send

VK2FGH WRU

Your system would then respond with

STATION IDENTIFICATION DE
VK2FGH (PETER)

If you had programmed the WRU buffer, your system might also add

PLEASE STAND BY . . .
++ OPERATOR ALERTED ++

or something similar.

If you wished to leave a special message your could put any code up to six letters long (followed by a space, of course) in the callsign storage buffer, and the special message in the WRU buffer. Only the stations aware of your code will be able to access the message.

Inbuilt pre-programmed buffers.
There are seven pre-programmed messages stored in your VZ200 terminal. Many of these are designed to insert your callsign automatically when called, to save you time and effort. These buffers and their enable commands are listed below:

Note: one row of text here is 32 characters. Thus it will only fill one half of a normal 64 character screen.

SHIFT C: Send — CQ
One row of CQs is sent along with your callsign

SHIFT A: Send — RYs
One row of RYs is sent.

SHIFT F: Send — QBF
Send 'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789'

SHIFT P: Send — over terminator.
The message 'PLEASE KK KK KK' is sent to terminate your call.

SHIFT I: Identify your station.
The message 'STATION IDENTIFICATION DE (callsign)' is sent. This is the same as is sent by the WRU mode.

SHIFT O: Send — Callsign.
Your callsign (as entered on power-up) is sent.

SHIFT D: Send — DE callsign.
As above except 'DE' is added to the start of your callsign.

Following are the commands to send the programmable buffers.

SHIFT Q: Send buffer #1

 4 of 5.

SHIFT W:	Send buffer #2
SHIFT E:	Send buffer #3
SHIFT R:	Send buffer #4
SHIFT T:	Send buffer #5
SHIFT 0:	Send WRU buffer (buffer #0)

At any time you may require to restart the system. This is useful if you wish to re-enter your callsign, or enter your own WRU code. To do this, type

SHIFT G

This exits the current mode and restarts at the callsign entry mode. You may now re-enter your callsign.

Printer Function.
Your VZ200 will also drive a line printer. You may enable or disable the printer mode using

SHIFT H

Once enabled, all text received or transmitted will be sent to the printer to be stored as 'hard copy'. Note: If you enable the printer but do not have a printer on-line, your system will not be affected and will ignore the enable mode. *But,* text will still be stored in the internal printer buffer until the buffer finally fills up.

The internal print buffer is only 64 characters long and is designed to hold characters only when the printer is busy printing. Because of this, any text received when the printer is not on-line but the print routine is enabled, will be truncated in the buffer. If you have the print mode enabled and don't want to print the text which has been stored in the internal print buffer, you may clear the buffer with the following command

SHIFT B

There will be times when a station does not terminate his contact with a CARRIAGE RETURN (CR). When this happens, you may find the last line of text does not get printed on the line printer. This is because many printers wait for a CR before printing the next line of text. By using the command

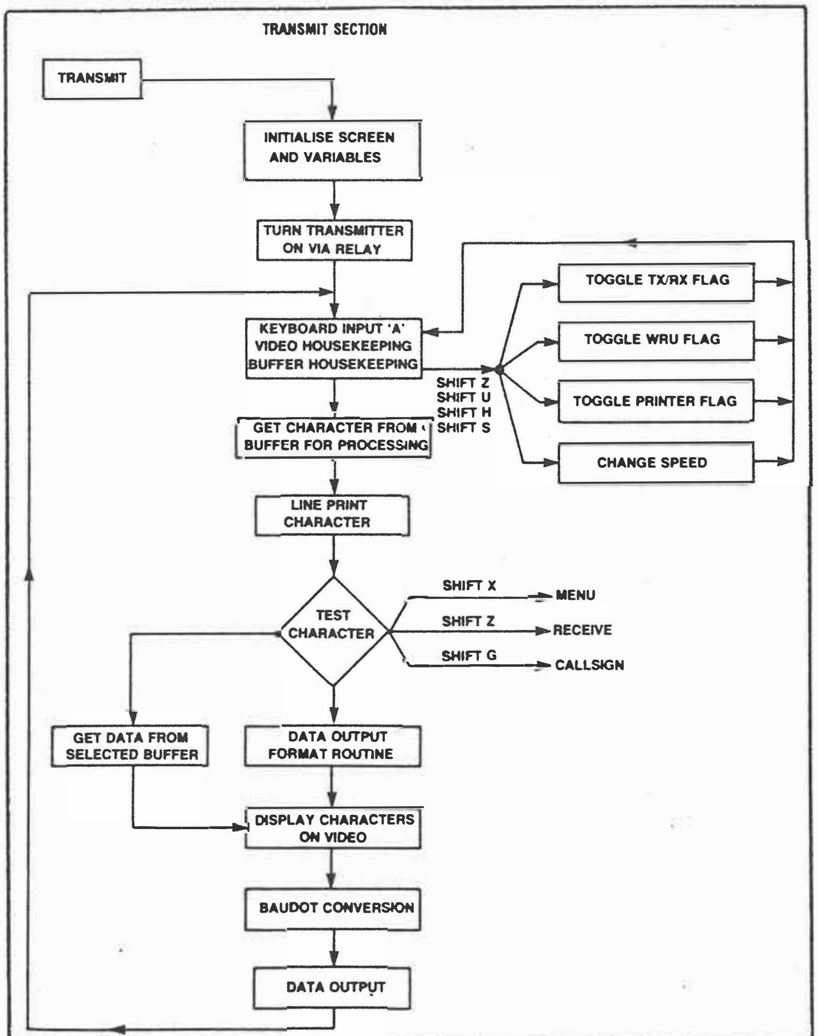
SHIFT <RETURN>

a carriage return will be inserted into the print buffer, thereby forcing it to print the last line. This can be done at any time to clear the printer's buffer, by forcing it to dump its contents onto paper.

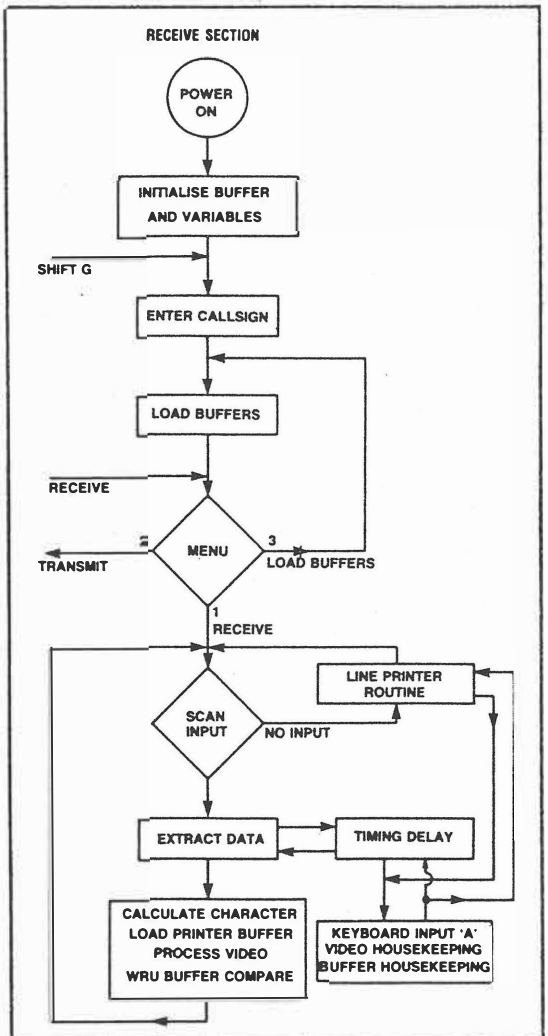
That concludes the main operation description. The rest will come with experience, as will normal RTTY operating procedures.

For further information on amateur RTTY, we suggest you contact *The Australian National Amateur Radio Teleprinter Society* at the following address:

The Secretary,
ANARTS,
PO Box 860,
Crows Nest NSW 2065



Flowchart for the TRANSMIT software.



Flowchart for the RECEIVE software.

5 of 5.

VZ-200 TERMINAL

With the addition of a low cost V.21 modem this project will get your Dick Smith VZ-200 talking to the world! Designed and developed by the DSE Research and Development team at North Ryde, the ETI-695 must be the cheapest way to get a 300 baud glass terminal going yet.

THE VZ-200 was very good 'value for money' when it was released by Dick Smith Electronics a few years ago. The last batch sold was heavily discounted and no doubt many were snapped up by ETI readers, especially RTTY enthusiasts after the ETI-756 RTTY adaptor appeared in Nov/Dec '84. This project extends the VZ's capability to operate as a 300 baud serial terminal. Although the VZ-200 is no longer available the unit will also work with the latest VZ-300 computer which has an improved keyboard.

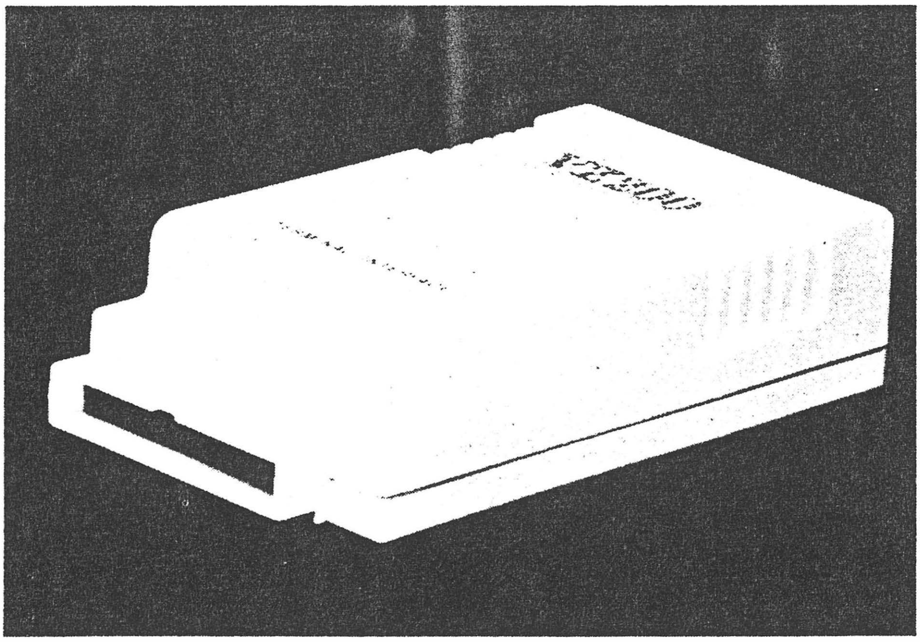
Construction

The pc board is designed to fit into a VZ expansion case which adds a professional finish to the project and is recommended. The case needs a bit of surgery to mount the DB-25S connector, so mark out the cut at the back of the 'top' half of the box (the larger piece). The connector sits flush with the lip of the half-case. Drill the two mounting holes for the DB-25S and screw it in with the 12 mm x 4BA screws and nuts.

Check over the pc board before commencing construction, look for broken tracks, bridges and undrilled holes. The prototype pc board has been tinned and had a couple of holes covered by the solder. These are best handled by heating the spot with a soldering iron and a bit of solder wick, if you try and force the component leads through such blocked holes you run the risk of lifting the copper away from the board and breaking bits off.

Start off by soldering in the ten wire links. One of them is near a mounting hole and should be bent around the hole to leave it uncovered, the other nine links should be straight and tight.

The 44-way edge connector can go in next. It mounts from the component side of the board (of course). The solder tails should be pushed through the board so that the bottom of the plastic part of the connector is flush with the copper side of the pc board. This is necessary to fit the finished pc board correctly into the case, so make sure the connector is aligned before soldering.



Some of the resistors mount on their ends. Be careful not to bend the leads too close to the resistor body to avoid breaking the leads off.

Solder in the capacitors before the diodes, since the two electrolytic caps are a wee bit close to diodes D4 and D5, which mount on their ends.

The two smaller transistors Q1 and Q2 can go in next, followed by Q3 which should be bent over if it is a BD139, as in the photograph. Solder the IC socket and the four ICs being careful to avoid solder bridges between the pins.

The three wires to the DB-25S connector were brought to the copper side of the pc board on the prototype; you may wire from the component side if you prefer before soldering.

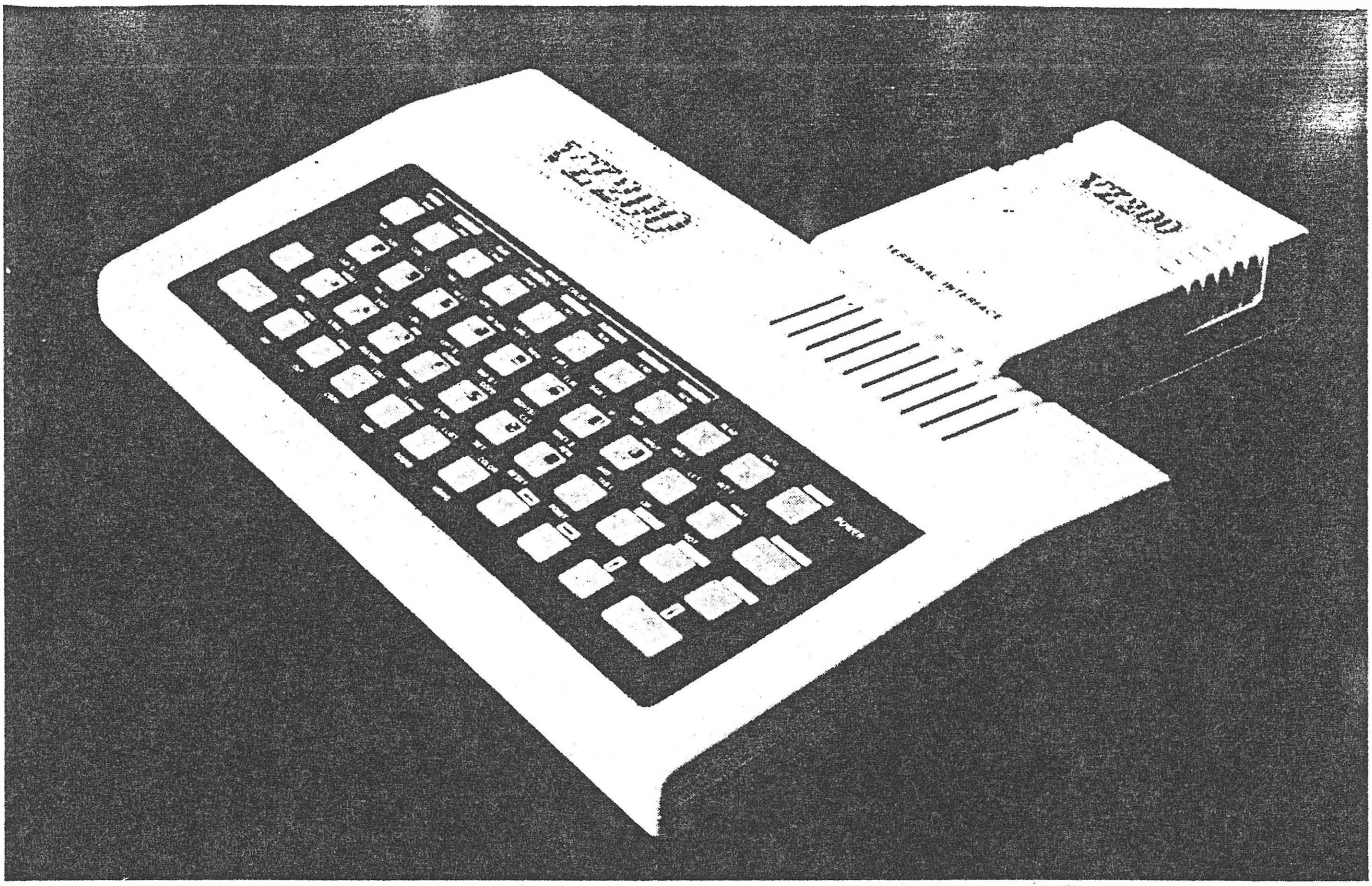
Place the bottom half of the case down and push the 44-way connector through the slot in the end with the copper side of the pc board uppermost. Align the two pc board holes with the mounting pillars and fit the top half of the case. Finish off by putting the case screws in and the project is ready to test.

Testing

Make sure your VZ-200 is operating properly before connecting the project. The interface plugs into the memory expansion port which is the largest on the back of the computer. Power should be switched off while inserting or removing the unit.

Testing is best done with a 300 baud terminal (or another computer emulating one) otherwise you will have to call a friend or bulletin board with a modem. To actually ▶

1 of 7.


VZ200
VZ300
TERMINAL INTERFACE


MP-0100-22-DW-4KR (SP)
3284
SP8338*
DM74LS74AN

2 of 7.

PARTS LIST — ETI-695

NOTE — A complete kit of parts can be obtained from your Dick Smith store.

Resistors.................all ¼ W, 5%
R1, 2, 3, 4, 10........4k7
R5, 12....................1k
R6..........................33k
R7, 11....................10k
R8, 9......................3k3
R13........................2k7

Capacitors
C1, 2.......................100n ceramic
C3, 4.......................10n polyester (greencap)
C5, 6.......................100µ 16 V RB electrolytic

Semiconductors
IC1.........................74LS138
IC2.........................2516 "VZRS" EPROM V1.5 or later
IC3.........................74LS74
IC4.........................74LS33
IC5.........................'555 timer
Q1 ....BC548
Q2 ....BC557
Q3 ....BD139 or BC639
D1, 2.......................1N60 Ge diodes
D3..........................1N914
D4, 5.......................1N4002

Miscellaneous
Printed circuit board "VZRS232"; VZ expansion case; 44-way female edge connector right angle pcb mounting; DB25S chassis socket; 2 x 12 mm 4BA screws and nuts; 24 pin DIP IC socket; tinned copper wire, hookup wire, solder, etc.

Price estimate: $49.95

SOFTWARE OPERATION

The VZ terminal interface is totally software based. This text is to serve as a functional description of the operation of this software.

The software resides in an EPROM on the interface board and maintains a data area in RAM at 8000 hex. In this data area are the flags and values used by the terminal software. At power-up these values are set to pre-defined values of 8 data bits, 1 stop bit and no parity. The unit is 300 baud only.

After the power-up sequence has been completed, the software goes into a loop waiting for keyboard input from the user. At this time the user can select one of seven menu options, these are:

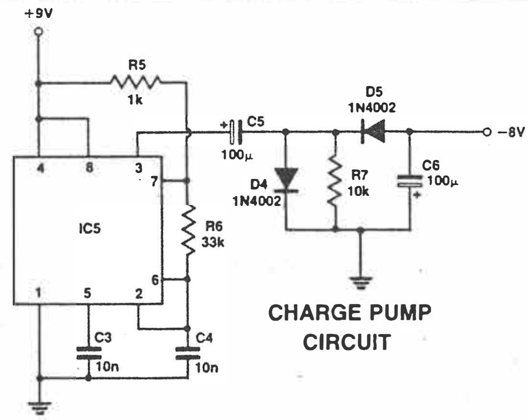
0) go to the terminal;
1) select full/half duplex;
2) toggle printer output on/off;
3) set number of data bits (7 or 8);
4) set number of stop bits (1 or 2);
5) set parity (odd, even or none);
6) set lf to cr option

If the user has selected one of the options 1-6, the appropriate action is taken and displayed on the screen. If option 0 is selected the software goes into terminal mode.

If the user selected option 0, the system begins looking for either keyboard input or incoming serial data. If a key has been pressed on the keyboard, then the software gets the value of that key, determines if it is a 'return to main menu' key (shift-x); if this is so it returns to the main menu, otherwise it sends the character to a routine that decodes it into bits and sends it serially to the interface hardware. It also adds start, stop and, optionally, parity bits. If the duplex option is set to half, it will echo to the screen as well.

If incoming serial data is found (by detecting a transition from a stop to a start bit), the software goes into a loop, reading bit seven of a port and encoding the incoming serial data bits into a byte, taking due consideration to the state of the start bit, stop bit(s) and optionally the parity bit. After a valid character is assembled it is sent to the screen and optionally to the printer.

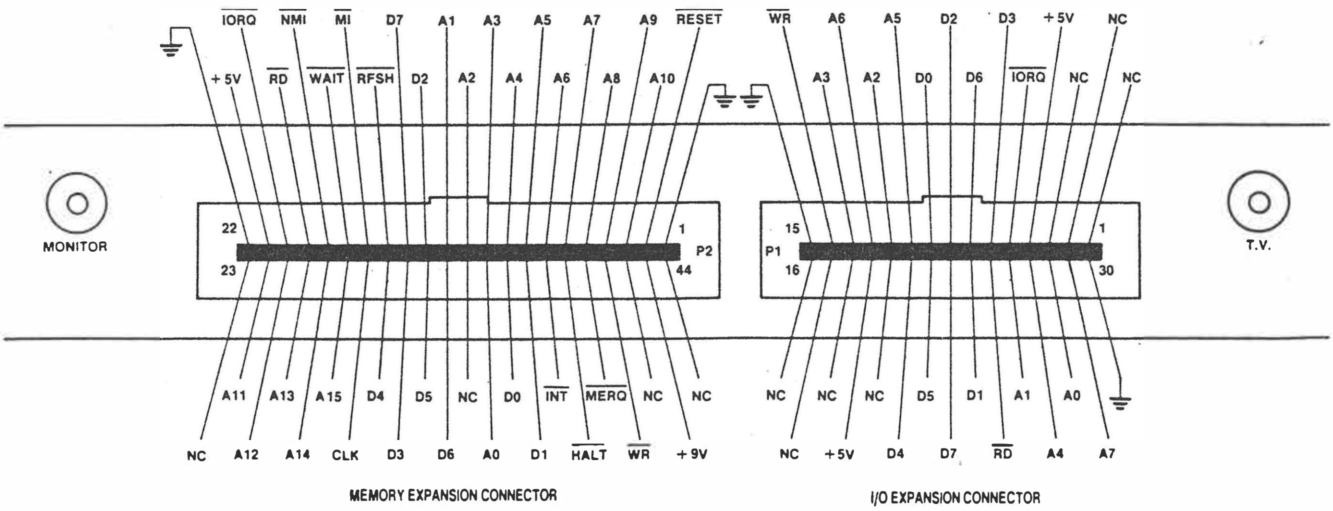
The terminal operation continues until it detects a shift-x key, at which time it returns to the main menu.



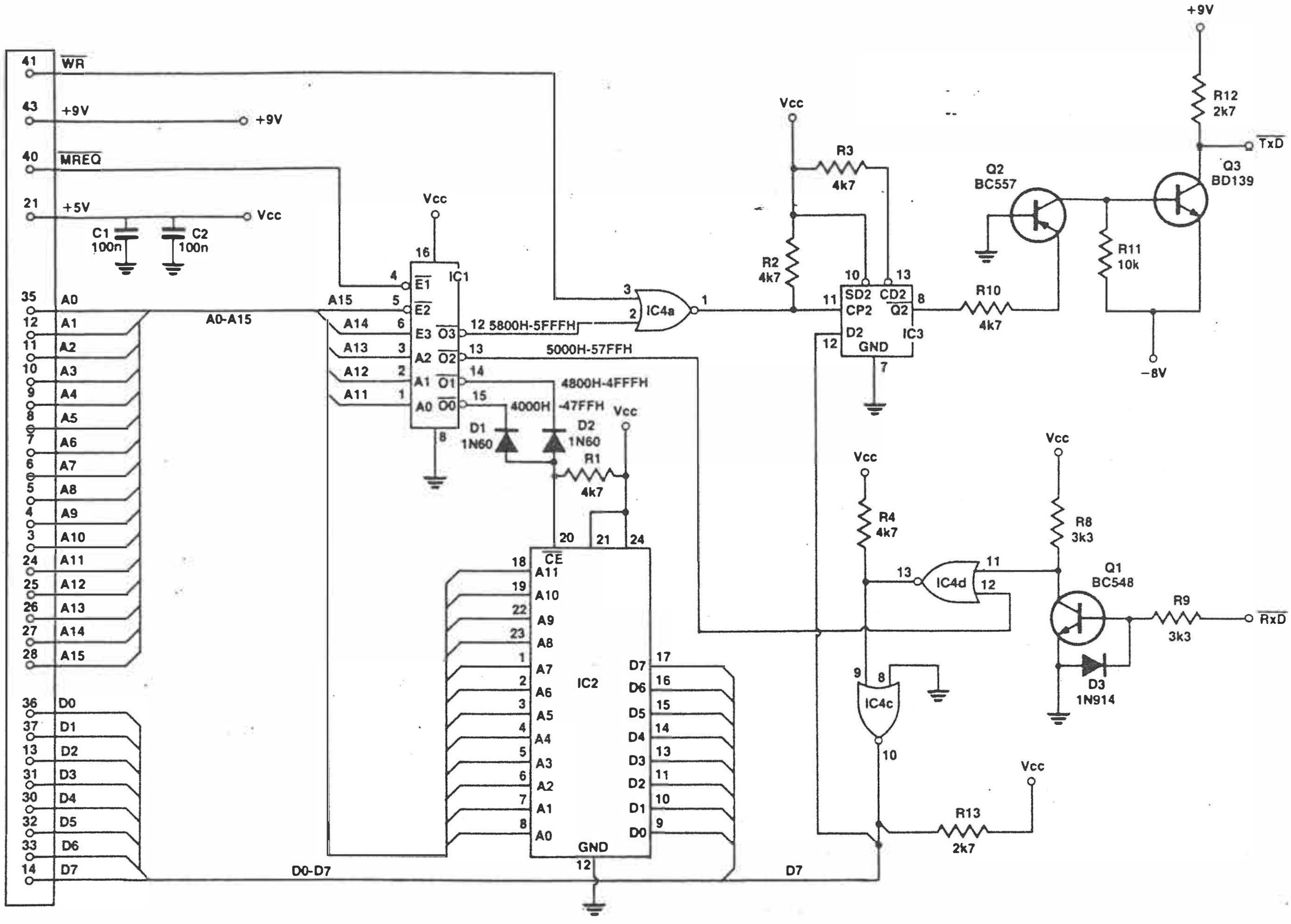
CHARGE PUMP CIRCUIT

ADDENDUM:
The edge connector connections
DO NOT agree with the
Tech. Manual. info.

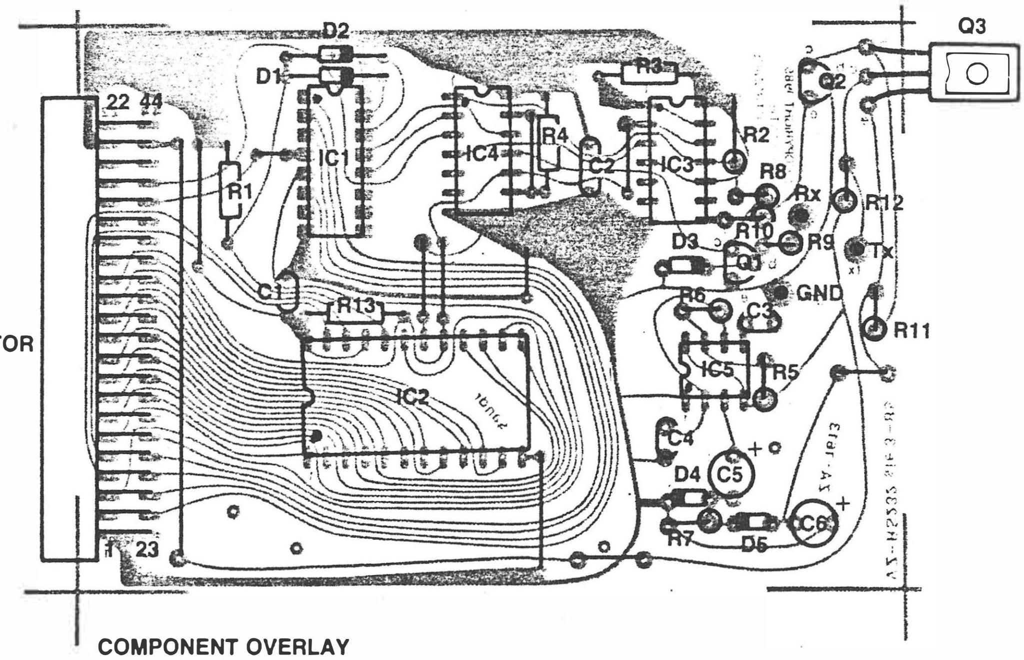
VZ-200 REAR PANEL LAYOUT



3 of 7.

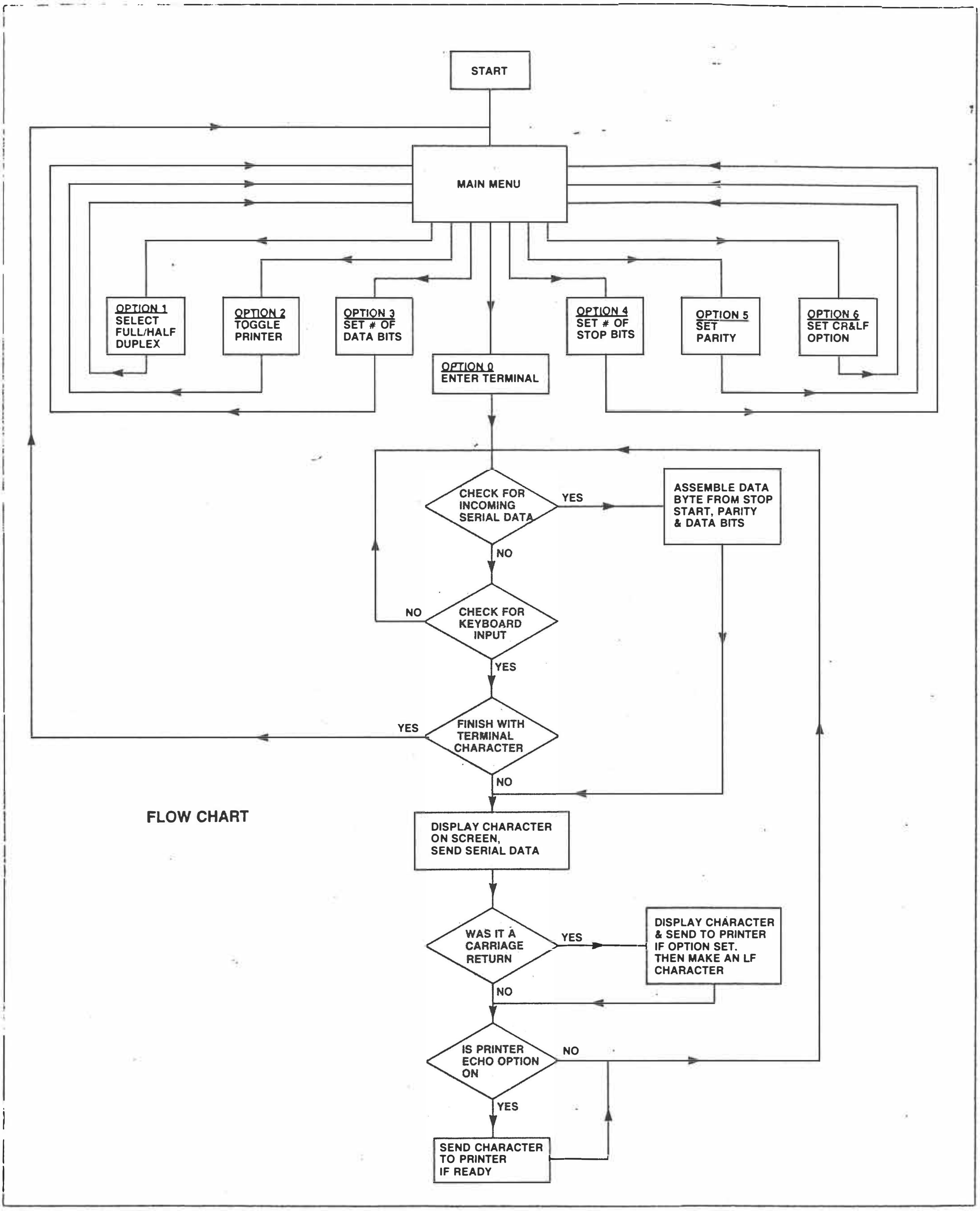


MAIN CIRCUIT DIAGRAM



COMPONENT OVERLAY

4 of 7.



FLOW CHART

5 of 7.

HOW IT WORKS — ETI-695

The terminal interface provides a Dick Smith VZ-200 or VZ-300 computer with the hardware and software necessary to emulate a simple 300 bit/s terminal. The software supports full or half duplex operation and has a printer echo option to record the conversation.

THE VZ-200 COMPUTER

The basic VZ-200 computer employs a Z80 microprocessor running at a clock speed of 3.58 MHz. Two 8K x 8 mask-programmed ROMs contain the Microsoft BASIC interpreter, while three 2K x 8 static RAMs provide program memory.

A 6847P-1 video controller chip and a further 2K x 8-bit static RAM form the heart of the computer's video section.

A simple software scanning scheme is used for the keyboard. The keys are arranged in eight rows, each of which can be pulled down to low logic level by diodes connected to the eight least significant address lines (A0-A7). The other sides of the keys are connected to six column lines, which are connected to six of the inputs of a gated octal buffer, and also to six pull-up resistors. The octal buffer's outputs are connected to the six least significant data lines of the processor (D0-D5).

Simplified decoding is used for selection of the various I/O devices in memory space. The memory address ranges occupied are as follows (in hexadecimal notation):

VZ-200 MEMORY MAP (WITH TERMINAL)

- 0000-1FFF basic ROM 0
- 2000-3FFF basic ROM 1
- 4000-47FF terminal EPROM
- 4800-4FFF spare space, can be used with 2532 EPROM
- 5000-57FF receive data, data on data bit 7
- 5800-5FFF transmit data latch, data sent on data bit 7
- 6000-67FF not used in terminal
- 6800-6FFF keyboard, cassette interface, speaker, VDC
- 7000-77FF video RAM
- 7800-8FFF inbuilt user RAM
- 9000-FFFF reserved for memory expansion modules

Note that due to the simplified addressing, the output latch serving the cassette output, speaker and video display controller effectively occupies all addresses from 6800-6FFF inclusive. Similarly the keyboard/cassette input buffer also occupies all of this address range, although the individual rows of keys effectively occupy discrete addresses.

For more information on the VZ-200 consult the *VZ-200 Technical Reference Manual* available from Dick Smith Electronics.

THE TERMINAL HARDWARE

The project connects to the VZ-200 through the memory expansion connector (P2) and is memory mapped.

IC1 decodes the Z-80's address lines to provide select signals for the EPROM IC2, the transmit latch IC3 and the receive data gates.

The incoming RS232 signal is converted from a −12/+12 volt signal to a TTL compatible signal by T1, thence to IC4 where it is gated with the 5000-57FF enable signal. If this enable signal is true (active low) the received data is inverted and fed to data bit D7 where it is read by the terminal software.

The outgoing TTL signal is sent from data bit D7 to IC3 where it is latched. The clock for IC3 is provided by gating the processor write enable with the 5800-5FFF output from IC1. The output from IC3 is level shifted by T2 and T3 to obtain an RS232 compatible signal. The negative voltage used by T3 is generated in a charge pump circuit based on IC5, a '555 timer.

SOURCE CODE

A complete documented source code listing of the software will be available on the Dick Smith Bulletin Board in the near future (according to Steven Engels of Dick Smith Electronics). The listing is too long to reproduce in the magazine. THE DSE-BBS is reached on: (02)887-2276 within Australia; +61 2 887-2276 on ISD.

The DSE-BBS is online 24 hours except on Fridays between 3 pm and 5.30 pm Eastern Standard Time.

TECHNICAL INQUIRIES

As the complete project including software was developed at DSE, all inquiries about the VZ-200 terminal project should be directed to Dick Smith Electronics.

communicate you have to enter the terminal mode from the menu by typing 0.

Providing the character length, parity and stop bits are identical you should have no trouble using the ETI-695 as a simple terminal.

We had some problems using the printer echo command with an Admate DP-80 printer using version 1.5 of the VZRS EPROM. This may be fixed in later versions, after our publication deadline. ●

HEXADECIMAL MACHINE CODE LISTING VZ-RS V1.5

```
ADDR   0  1  2  3   4  5  6  7   8  9  A  B   C  D  E  F

0100:  AA 55 E7 18  C3 84 41 4F  4E 20 4F 46  46 46 55 4C
0110:  4C 48 41 4C  46 0C 56 5A  2D 32 30 30  2F 33 30 30
0120:  20 52 53 2D  32 33 32 20  2D 20 56 45  52 53 49 4F
0130:  4E 20 31 2E  35 0D 28 43  29 20 31 39  38 35 20 44
0140:  49 43 4B 20  53 4D 49 54  48 20 45 4C  45 43 54 52
0150:  4F 4E 49 43  53 0D 2D 2D  2D 2D 2D 2D  2D 2D 2D 2D
0160:  2D 2D 2D 2D  2D 2D 2D 2D  2D 2D 2D 2D  2D 2D 2D 2D
0170:  2D 2D 2D 2D  2D 0D 30 5D  20 45 4E 54  45 52 20 54
0180:  45 52 4D 49  4E 41 4C 0D  31 5D 20 46  55 4C 4C 2F
0190:  48 41 4C 46  20 44 55 50  4C 45 58 3A  00 46 55 4C
01A0:  4C 0D 32 5D  20 54 4F 47  47 4C 45 20  50 52 49 4E
01B0:  54 45 52 20  20 3A 4F 46  46 0D 33 5D  20 53 45 54
01C0:  20 23 20 44  41 54 41 20  42 49 54 53  20 3A 38 20
01D0:  20 0D 34 5D  20 53 45 54  20 23 20 53  54 4F 50 20
01E0:  42 49 54 53  20 3A 31 20  20 0D 35 5D  20 53 45 54
01F0:  20 50 41 52  49 54 59 20  20 20 20 20  20 3A 4E 20
```

```
ADDR   0  1  2  3   4  5  6  7   8  9  A  B   C  D  E  F

0200:  20 0D 36 5D  20 41 44 44  20 4C 46 20  54 4F 20 43
0210:  52 20 20 20  20 3A 4F 46  46 0D 0D 2A  2A 2A 2A 2A
0220:  2A 2A 20 57  48 45 4E 20  49 4E 20 54  45 52 4D 49
0230:  4E 41 4C 20  2A 2A 2A 2A  2A 2A 2A 2A  20 20 53 48
0240:  49 46 54 20  2D 20 58 20  54 4F 20 45  58 49 54 20
0250:  54 45 52 4D  49 4E 41 4C  20 20 2A 2A  2A 2A 2A 2A
0260:  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 2A
0270:  2A 2A 2A 2A  2A 2A 2A 2A  2A 2A 2A 00  00 00 00 00
0280:  00 00 00 00  F3 31 00 90  21 9D 40 11  00 80 01 E7
0290:  00 ED B0 3A  E1 80 F5 3E  01 32 E1 80  21 15 40 CD
02A0:  4D 43 21 00  80 CD 4D 43  F1 32 E1 80  CD 50 34 21
02B0:  93 41 E5 CD  66 44 B7 28  FA D6 30 38  F6 FE 07 30
02C0:  F2 21 CE 41  87 5F 16 00  19 5E 23 56  EB E9 56 42
02D0:  DC 41 F7 41  38 42 47 42  0B 42 01 42  3A E0 80 B7
02E0:  3E 01 21 11  40 28 04 AF  21 0D 40 32  E0 80 11 00
02F0:  80 01 04 00  ED B0 C9 21  DF 80 11 19  80 CD 21 42
```

```
ADDR   0  1  2  3   4  5  6  7   8  9  A  B   C  D  E  F

0300:  C9 21 E1 80  11 79 80 CD  21 42 C9 3A  61 80 FE 4E
0310:  0E 45 28 08  FE 45 0E 4F  28 02 0E 4E  79 32 61 80
0320:  C9 7E B7 3E  01 28 01 AF  77 21 0A 40  B7 28 03 21
0330:  07 40 01 03  00 ED B0 C9  3A 31 80 FE  37 3E 38 28
0340:  02 3E 37 32  31 80 C9 3A  49 80 FE 31  3E 32 28 02
0350:  3E 31 32 49  80 C9 CD 56  43 CD 1D 43  20 0E CD 8F
0360:  42 CD 6E 43  4F 3A DF 80  B7 C4 53 44  CD 66 44 B7
0370:  28 E7 FE 65  C8 F5 CD C6  42 F1 4F 3A  E0 80 B7 79
0380:  28 D7 CD 6E  43 4F 3A DF  80 B7 C4 53  44 18 CA CD
0390:  2E 43 CD 23  43 3A 31 80  D6 30 5F 06  08 0E 00 78
03A0:  FE 01 20 07  7B FE 08 28  0E 18 0E 3A  00 50 E6 80
03B0:  B1 4F CB 39  CD 23 43 10  E6 CD 3A 43  3A 61 80 FE
03C0:  4E C4 23 43  79 C9 F5 3E  FF 32 00 58  CD 23 43 3A
03D0:  31 80 D6 30  47 F1 F5 4F  CB 39 3E 00  38 02 3E 80
03E0:  32 00 58 CD  23 43 10 F0  F1 4F 3A 61  80 FE 4E 28
03F0:  1C 79 B7 3A  61 80 E2 03  43 FE 45 3E  80 28 08 3E
```

continued ▶

6of 7.

MACHINE CODE LISTING CONTINUED

ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0400:	00	18	04	FE	4F	18	F4	32	00	58	CD	23	43	3A	49	80
0410:	D6	30	47	AF	32	00	58	CD	23	43	10	F7	C9	3A	00	50
0420:	CB	7F	C9	F5	C5	CD	2E	43	CD	2E	43	C1	F1	C9	C5	3E
0430:	23	06	0B	10	FE	3D	20	F9	C1	C9	CD	2E	43	C5	3E	22
0440:	18	EF	F5	C5	01	FF	4F	CD	60	00	C1	F1	C9	7E	B7	C8
0450:	CD	6E	43	23	18	F7	21	00	70	22	E5	80	11	01	70	01
0460:	FF	01	36	60	ED	B0	AF	32	E4	80	32	00	68	C9	F5	E5
0470:	C5	D5	CD	7A	43	D1	C1	E1	F1	C9	ED	5B	E5	80	FE	0C
0480:	28	D4	FE	0D	28	7E	FE	08	28	35	FE	09	28	16	FE	0A
0490:	28	4A	FE	07	CA	50	34	CB	7F	20	08	FE	20	F8	CD	5D
04A0:	44	CB	F7	12	13	ED	53	E5	80	3A	E4	80	3C	32	E4	80
04B0:	FE	20	F8	CD	F3	43	3A	DF	80	B7	C8	CD	49	44	C9	3A
04C0:	E4	80	B7	28	0A	3D	32	E4	80	1B	ED	53	E5	80	C9	E5
04D0:	21	00	70	B7	ED	52	E1	C8	3E	1F	18	EA	3A	E4	80	4F
04E0:	06	00	C5	CD	F3	43	C1	EB	09	EB	ED	53	E5	80	79	32
04F0:	E4	80	C9	3A	E1	80	F5	3E	01	32	E1	80	CD	04	44	F1

ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0500:	32	E1	80	C9	E5	EB	3A	E4	80	5F	16	00	B7	ED	52	3A
0510:	E1	80	B7	28	04	11	20	00	19	EB	21	00	72	B7	ED	52
0520:	28	0A	ED	53	E5	80	AF	32	E4	80	E1	C9	21	20	70	11
0530:	00	70	01	E0	01	ED	B0	21	E0	71	11	E1	71	01	1F	00
0540:	36	60	ED	B0	11	E0	71	18	D9	CD	C4	05	CB	47	C0	CD
0550:	E2	3A	C9	CD	C4	05	CB	47	C0	79	C3	8D	05	FE	61	D8
0560:	FE	7B	D0	E6	5F	C9	21	FE	68	0E	08	06	06	7E	F6	04
0570:	1F	30	58	10	FB	CB	05	0D	20	F1	06	04	21	DF	68	7E
0580:	CB	57	28	3D	CB	05	7E	CB	57	28	3A	CB	05	CB	57	28
0590:	38	CB	05	CB	05	7E	CB	57	28	11	CB	05	7E	CB	57	28
05A0:	11	3E	FF	32	E2	80	AF	32	E3	80	C9	3A	E3	80	CB	D7
05B0:	18	05	3A	E3	80	CB	CF	32	E3	80	3E	FF	32	E2	80	AF
05C0:	C9	0E	03	18	06	0E	02	18	02	0E	01	21	05	45	1E	00
05D0:	3A	E3	80	CB	57	28	04	1E	60	18	06	CB	4F	28	02	1E
05E0:	30	3E	08	91	4F	3E	06	90	47	CD	FA	44	83	06	00	4F
05F0:	09	7E	21	E2	80	BE	28	C7	77	C9	AF	B9	28	05	C6	06

ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0600:	0D	20	FB	80	C9	54	57	20	45	51	52	47	53	20	44	41
0610:	46	42	58	20	43	5A	56	35	32	20	33	31	34	4E	2E	20
0620:	2C	20	4D	36	39	2D	38	30	37	59	4F	0D	49	50	55	48
0630:	4C	3A	4B	3B	4A	00	00	00	00	00	00	00	00	00	00	00
0640:	00	00	65	00	00	00	00	25	22	20	23	21	24	00	3E	00
0650:	3C	00	5C	26	29	3D	28	40	00	00	00	00	00	00	00	00
0660:	3F	2A	2F	2B	00	14	17	00	05	11	12	07	13	00	00	01
0670:	06	02	18	00	03	1A	16	00	00	00	00	00	00	0E	00	00
0680:	00	00	0D	00	00	00	00	00	00	19	0F	00	09	10	15	08
0690:	0C	00	0B	00	0A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
06A0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
06B0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
06C0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
06D0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
06E0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
06F0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0700:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0710:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0720:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0730:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0740:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0750:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0760:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0770:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0780:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0790:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
07A0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
07B0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
07C0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
07D0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
07E0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
07F0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

7 of 7.

VZ serial terminal

Connecting a modem to the popular VZ-200 and VZ-300 computers is now possible with the recently released Serial Terminal kit from Dick Smith Electronics Pty Ltd. The kit is both inexpensive and easy to assemble.

When plugged into a VZ series computer, the serial interface provides all hardware and software necessary to emulate a simple 300 baud terminal with full or half duplex operation. It also has a printer echo option to record the conversation.

The device incorporates facilities to set serial data format, add an optional auto line feed on carriage return, and to dump all communications to a parallel printer if one is connected.

For further information contact Dick Smith Electronics Pty Ltd, PO Box 321, North Ryde, 2113. Telephone (02) 888 3200.

Rec'd Jamie Perry 2/7/86

```
;
title VZ-200/300 RS-232 terminal rom software
subttl Version 1.5 last revised 11th June 1985 at 2am in the morning !!!! SLE.
;
;VZ-200/300 RS-232/TERMINAL cartridge pack
;
;This source code and the ROM program it generates is copyright (c) 1985
;by Dick Smith Electronics Pty Ltd and may not be used for any commercial
;gain in any form whatsoever. The Dick Smith Electronics copyright message
;must not be removed from this source code or from the ROM program. All rights
;commercial and otherwise are retained by the copyright holder
;
;Permission is granted for constructors of the VZ 200/300 terminal
;interface to use these routines for construction of said project
;and for instructional use ONLY.
;
;This source code (and all routines contained herein) shall remain the
;property of Dick Smith Electronics Pty Ltd. Any variation of these copyright
;notices must be in writing from Dick Smith Electronics Pty Ltd.
;
;
;This source code file is set up to be assembled using Microsoft's M80
;macro assembler and was developed on a DSE Bondwell 14 portable computer.
;
;
crtdge  equ     04000h
stack   equ     09000h
vzdelay equ     060h            ;rom delay routine, delay value in bc
beep    equ     03450h          ;beep routine
pcrlf1  equ     03ae2h          ;send cr and lf to printer
list    equ     058dh           ;printer driver
listst  equ     05c4h           ;printer status, bit 0 of A=0 if ready
chrot   equ     033ah           ;character output routine
exitkey equ     101             ;shift 'X' is exit from term
outadr  equ     05800h          ;transmit latch address
inadr   equ     05000h          ;input address
toplin  equ     28672           ;top line of screen
nolin   equ     480             ;(number of lines - 1) * 32
;
;ascii definitions.
;
home    equ     28
cr      equ     13
```

```
esc     equ     27
bell    equ     7
ff      equ     12
lf      equ     10
bs      equ     8
ht      equ     9
;
;Now define our storage area, where basic text usually starts
;
dumsg   equ     08000h          ;part of the signon message that changes
pntmsg  equ     dumsg+25        ;printer on/off message
dbits   equ     pntmsg+24       ;dbits message
sbits   equ     dbits+24        ;sbits message
parmsg  equ     sbits+24        ;parity message
plmsg   equ     parmsg+24       ;strip parity message
pntflg  equ     plmsg+count3    ;printer on/off flag
dupflg  equ     pntflg+1        ;duplex flag, 0=full
plflg   equ     dupflg+1        ;add lf to cr flag
char    equ     plflg+1         ;character found
flag    equ     char+1          ;keyboard flag
curpos  equ     flag+1          ;column counter for output routine
cursor  equ     curpos+1        ;cursor location
bufend  equ     09000h          ;to accomodate VZ-200/300
;
        .z80
        aseg
        org     0100h
        .phase  crtdge          ;place that it will run
;
begadr  equ     $
;
        defb    0aah            ;mask for rom pack installation recognition
        defb    055h            ;checked at power-up by the basic roms
        defb    0e7h
        defb    018h
;
        jp      start
;
;A few messages....
;
onmsg:  defb    'ON '
offmsg: defb    'OFF'
fullmsg:defb    'FULL'
halfmsg:defb    'HALF'
;
signon: defb    ff
        defb    'VZ-200/300 RS-232 - VERSION 1.5',cr
        defb    '(C) 1985 DICK SMITH ELECTRONICS',cr
        defb    '-------------------------------',cr
        defb    '0] ENTER TERMINAL',cr
        defb    '1] FULL/HALF DUPLEX:',0
;
ivals:  defb    'FULL',cr
        defb    '2] TOGGLE PRINTER  :'
        defb    'OFF',cr
        defb    '3] SET # DATA BITS :'
        defb    '8  ',cr
        defb    '4] SET # STOP BITS :'
        defb    '1  ',cr
        defb    '5] SET PARITY      :'
        defb    'N  ',cr
        defb    '6] ADD LF TO CR    :'
count1  equ     $
        defb    'OFF'
        defb    cr,cr
        defb    '******* WHEN IN TERMINAL *******'
```

```
        defb    '*  SHIFT - X TO EXIT TERMINAL  *'
        defb    '*********************************',0
count2  equ     $
count3  defl    count2-count1
ivals2  equ     $

        defb    0       ;printer on/off flag
        defb    0       ;duplex flag
        defb    0       ;add lf to cr flag
        defb    0       ;where characters are placed when recvd from the keybrd
        defb    0       ;a flag for the keyboard driver
        defb    0       ;column counter
        defw    0       ;cursor position
;
ivalend equ     $
;
start:  di                      ;get rid of the 6847 interrupt
        ld      sp,stack        ;make a stack
        ld      hl,ivals        ;point to initial values
        ld      de,dumsg        ;where to put them
        ld      bc,ivalend-ivals ;number to put
        ldir                    ;init the system
start1: ld      a,(plflg)       ;get crlf flag
        push    af              ;save it
        ld      a,1             ;we want crlf on signon
        ld      (plflg),a
        ld      hl,signon       ;point to signon
        call    msgout
        ld      hl,dumsg
        call    msgout
        pop     af              ;restore crlf flag
        ld      (plflg),a       ;and save it here
        call    beep            ;make a noise
        ld      hl,start1       ;set return address
        push    hl              ;put it on the stack
kget:   call    getkey          ;get a key
        or      a
        jr      z,kget          ;loop till key ready
        sub     '0'             ;make it 0-10
        jr      c,kget          ;continue if error
        cp      6+1
        jr      nc,kget         ;continue if error
        ld      hl,cmdtab       ;point to command table address
        add     a,a             ;make it *2
        ld      e,a             ;put it in de
        ld      d,0
        add     hl,de           ;now hl points to correct table entry
        ld      e,(hl)          ;get lsb
        inc     hl
        ld      d,(hl)          ;get msb, now hl points to correct address
        ex      de,hl
        jp      (hl)            ;go to that address
;
;table of addresses for the command table options
;
cmdtab: defw    term            ;actual terminal
        defw    setfh           ;set full/half duplex
        defw    prnton          ;turn printer on/off
        defw    sdbits          ;set # of data bits
        defw    ssbits          ;set # of stop bits
        defw    spar            ;set parity
        defw    addlf           ;add on lf to cr
;
;set full or half duplex
;
setfh:  ld      a,(dupflg)      ;check current
```

```
        or      a               ;check for full already
        ld      a,1             ;make half just in case
        ld      hl,halfmsg
        jr      z,setfhl
        xor     a               ;else make full
        ld      hl,fullmsg
setfhl: ld      (dupflg),a
        ld      de,dumsg        ;where to go to
        ld      bc,4            ;move it
        ldir
        ret
;
;toggle printer
;
prnton: ld      hl,pntflg       ;check printer flag
        ld      de,pntmsg
        call    toggle
        ret
;
;add lf to cr flag
;
addlf:  ld      hl,plflg
        ld      de,plmsg
        call    toggle
        ret
;
;set parity
;
spar:   ld      a,(parmsg)      ;get parity message
        cp      'N'             ;none ?
        ld      c,'E'           ;make it an even then
        jr      z,stparl        ;save this one if z flag set
        cp      'E'
        ld      c,'O'
        jr      z,stparl        ;if even, make an odd
        ld      c,'N'           ;must be odd, make none
stparl: ld      a,c
        ld      (parmsg),a
        ret
;
toggle: ld      a,(hl)          ;get current flag value
        or      a               ;is it zero
        ld      a,1             ;make it one if so
        jr      z,toggl
        xor     a               ;else make it a zero
toggl:  ld      (hl),a          ;save new value
        ld      hl,offmsg
        or      a
        jr      z,moveitl
        ld      hl,onmsg
moveitl:ld      bc,3
moveit: ldir
        ret
;
;set databits
;
sdbits: ld      a,(dbits)       ;get current value
        cp      '7'             ;seven ?
        ld      a,'8'           ;just in case
        jr      z,sdbl
        ld      a,'7'
sdbl:   ld      (dbits),a
        ret
;
;set stop bits
;
```

```
ssbits: ld      a,(sbits)       ;get current value
        cp      '1'             ;seven ?
        ld      a,'2'           ;just in case
        jr      z,ssb2          ;if z, must be 8, make it 7
        ld      a,'1'
ssb2:   ld      (sbits),a
        ret
;
term:   call    clrscr          ;clear the screen
tryser: call    chkstr          ;check for incoming serial stuff
        jr      nz,term1        ;no-look for keyboard input then
        call    inchr           ;stuff there, get it
        call    chrout          ;and print it on the screen
        ld      c,a             ;put char in correct register for list routine
        ld      a,(pntflg)
        or      a
        call    nz,list1
term1:  call    getkey          ;see if user has typed anything
        or      a
        jr      z,tryser        ;no, try the serial instead
        cp      exitkey         ;see if a return to main loop key
        ret     z               ;return if so
        push    af              ;save char
        call    outchr          ;send to RS-232
        pop     af              ;restore for screen output
        ld      c,a
        ld      a,(dupflg)      ;check duplex first
        or      a
        ld      a,c
        jr      z,tryser        ;jump if full duplex
        call    chrout          ;and screen as well if 1/2 duplex
        ld      c,a             ;put char in correct register for list routine
        ld      a,(pntflg)
        or      a ;
        call    nz,list1
        jr      tryser
;
;get a character from the board
;
;incoming char must have been checked by a call to chkstr
;
;This routine is time critical and as i took,be a long time to get it right,
;i would suggest that it not be touched.
;
inchr:  call    del3a           ;wait for 1/2 bit time, half way thru start bit
        call    del300          ;and 1 bit time, half way thru data bit 0
        ld      a,(dbits)       ;get data bit value
        sub     '0'             ;remove ascii offset, 8 for 8 bits 7 for 7
        ld      e,a             ;and put the calculated data bit value in e
        ld      b,8             ;must always do 8 shifts to get bit 0 correct
        ld      c,0             ;accumulator for our serial bit stream
inchr1: ld      a,b             ;check for last loop
        cp      1               ;are we nearly finished (we are if seven bits)
        jr      nz,inchr2       ;no-continue
        ld      a,e             ;see if 7 or 8 data bits required
        cp      8               ;if e=8, 8 bits are needed and one further loop
        jr      z,inchr3
        jr      inchr4          ;finish up if seven
;
inchr2: ld      a,(inadr)       ;get the entered bit
        and     10000000b       ;mask off invalid bits
        or      c               ;mask on our accumulated value
        ld      c,a             ;and save it here
        srl     c               ;and shift it into the next bit position
        call    del300          ;delay one 300 baud bit time
inchr3: djnz    inchr1          ;loop for all data bits
```

```
inchr4: call    del302          ;wait till end of data bit
        ld      a,(parmsg)      ;check if parity delay needed
        cp      'N'
        call    nz,del300
        ld      a,c             ;get the char back
        ret
;
;output char
;
outchr: push    af              ;save the char
        ld      a,255           ;make a start bit
        ld      (outadr),a      ;send it
        call    del300          ;and do a delay for 1 bit time
        ld      a,(dbits)       ;get # of data bits
        sub     '0'
        ld      b,a             ;and put the calculated data bit value in b
        pop     af
        push    af
        ld      c,a
snd1:   srl     c               ;rotate bit 0 of c into carry flag
        ld      a,00000000b     ;in case of carry
        jr      c,snd2          ;send if bit was 1
        ld      a,10000000b     ;make it one
snd2:   ld      (outadr),a      ;send the char
        call    del300          ;delay for one bit time
        djnz    snd1
        pop     af              ;restore char that was sent
        ld      c,a             ;save it in c
        ld      a,(parmsg)      ;check for parity
        cp      'N'
        jr      z,nopar0        ;z if no parity
        ld      a,c             ;get back restored character
        or      a               ;set parity flags
        ld      a,(parmsg)      ;and get parity type for later testing
        jp      po,oddpar       ;jump if parity of the byte is odd
        cp      'E'             ;was it supposed to be even
tstpar: ld      a,10000000b
        jr      z,sndpar
        ld      a,00000000b
        jr      sndpar
;
oddpar: cp      'O'             ;set z flag if it was supposed to be odd
        jr      tstpar

sndpar: ld      (outadr),a
        call    del300
nopar0: ld      a,(sbits)       ;get # of stop bits
        sub     '0'             ;make it hex 1 or 2
        ld      b,a
sbdel1: xor     a               ;make a stop bit
        ld      (outadr),a
        call    del300          ;delay for middle of stop bit
        djnz    sbdel1
        ret
;
;check for a character coming in
;
;RX provides an active low input to the address buffer
;
;i.e. returned value is nz if a char is ready, z if not
;
chkstr: ld      a,(inadr)       ;check input address
        bit     7,a
        ret
;
;These delay routines provide a correct mark to space ratio for checking
```

```
;and sending the serial bit stream and are VERY VERY VERY critical.
;if you send a series of 'U' (a character with the same mark to space ratio)
;characters and look at the output of the interface on a CRO, you will
;see what i mean.
;
;
del300: push    af
        push    bc
        call    del3a
        call    del3a
        pop     bc
        pop     af
        ret
;
del3a:  push    bc
        ld      a,215/6         ;300 baud delay
loop0:  ld      b,11
loop1:  djnz    loop1
        dec     a
        jr      nz,loop0
        pop     bc
        ret
;
del302: call    del3a           ;one 1/2 300 baud delay
        push    bc              ;save this
        ld      a,205/6         ;nex timing constant
        jr      loop0
;
kdelay: push    af
        push    bc
        ld      bc,04fffh       ;delay value
        call    vzdelay         ;delay it
        pop     bc              ;get registers back
        pop     af
        ret
;
msgout: ld      a,(hl)
        or      a
        ret     z
        call    chrout
        inc     hl
        jr      msgout
;
;Character output routine
;
clrscr: ld      hl,toplin
        ld      (cursor),hl
        ld      de,toplin+1
        ld      bc,nolin+31
        ld      (hl),96
        ldir
        xor     a
        ld      (curpos),a
        ld      (26624),a
        ret
;
chrout: push    af
        push    hl
        push    bc
        push    de
        call    pntit
        pop     de
        pop     bc
        pop     hl
        pop     af
        ret
```

```
;
pntit:  ld      de,(cursor)
        cp      ff
        jr      z,clrscr
        cp      cr
        jr      z,crt           ;do a cr
        cp      bs
        jr      z,bsp
        cp      ht
        jr      z,fsp
        cp      lf
        jr      z,lfd
        cp      bell
        jp      z,beep
        bit     7,a
        jr      nz,grap
        cp      32              ;if not recognised control char, return
        ret     m
        call    fold            ;fold lower to upper
        set     6,a             ;change non-alphabetical char
                                ;to black with white background
;
grap:   ld      (de),a
fsp:    inc     de
        ld      (cursor),de
        ld      a,(curpos)
        inc     a
        ld      (curpos),a
        cp      32
        ret     m
        call    docrlf
        ld      a,(pntflg)      ;printer on ?
        or      a
        ret     z               ;return if not
        call    pcrlf
        ret
;
bsp:    ld      a,(curpos)
        or      a
        jr      z,bsp1
        dec     a
bsp2:   ld      (curpos),a
        dec     de
        ld      (cursor),de
        ret
;
bsp1:   push    hl
        ld      hl,toplin
        or      a
        sbc     hl,de
        pop     hl
        ret     z
        ld      a,31
        jr      bsp2
;
lfd:    ld      a,(curpos)
        ld      c,a
        ld      b,0
        push    bc
        call    docrlf
        pop     bc
        ex      de,hl
        add     hl,bc
        ex      de,hl
        ld      (cursor),de
        ld      a,c
```

```
        ld      (curpos),a
        ret
;
docrlf: ld      a,(plflg)       ;get current setting
        push    af              ;save it
        ld      a,1             ;make it so it does add on lf
        ld      (plflg),a
        call    crt             ;do it
        pop     af              ;get back original flag
        ld      (plflg),a       ;and save it again
        ret
;
crt:    push    hl
        ex      de,hl
        ld      a,(curpos)
        ld      e,a
        ld      d,0
        or      a
        sbc     hl,de
        ld      a,(plflg)       ;do we want to add on an lf to a cr
        or      a               ;if zero we don't
        jr      z,nolf
        ld      de,32           ;add on an lf if cr
        add     hl,de
nolf:   ex      de,hl
        ld      hl,29184
        or      a
        sbc     hl,de           ;check for end of screen
        jr      z,scroll        ;scroll if so
write4: ld      (cursor),de
        xor     a
        ld      (curpos),a
        pop     hl
        ret
;
scroll: ld      hl,toplin+32
        ld      de,toplin
        ld      bc,nolin
        ldir
        ld      hl,29152
        ld      de,29153
        ld      bc,31
        ld      (hl),96
        ldir
        ld      de,29152
        jr      write4
;
pcrlf:  call    listst          ;check printer status
        bit     0,a
        ret     nz              ;return if printer not ready
        call    pcrlf1          ;send crlf to printer if printer is ready
        ret
;
;
listl:  call    listst
        bit     0,a
        ret     nz              ;return if printer not ready
        ld      a,c
        jp      list            ;list it and return
;
fold:   cp      'a'
        ret     c
        cp      'z'+1
        ret     nc
        and     05fh            ;make upper if lower
        ret
```

```
;
;Keyboard driver for VZ-200/300
;
;This keyboard drive is (c) Dick Smith Electronics and (c) 1982,1983,1984
;Video Technology HK Ltd.
;
row1    equ     068feh
row2    equ     068dfh
;
getkey: ld      hl,row1             ;point to first row
        ld      c,8                 ;row counter
scan1:  ld      b,6                 ;column counter
        ld      a,(hl)              ;get first key
        or      4                   ;mask out bit 2
rot:    rra                         ;rotate bits
        jr      nc,found            ;exit if key pressed
        djnz    rot                 ;else try next bit
        rlc     l                   ;get next address
        dec     c                   ;dec row counter
        jr      nz,scan1            ;try next row
        ld      b,4                 ;get col counter
        ld      hl,row2             ;get next address
        ld      a,(hl)              ;read key
        bit     2,a                 ;test bit 2 for keypress
        jr      z,minus             ;exit if key pressed
        rlc     l                   ;get next address
        ld      a,(hl)              ;read key
        bit     2,a                 ;test for keypress
        jr      z,carret            ;exit if cr key pressed
        rlc     l                   ;get next addr
        bit     2,a
        jr      z,colon             ;test if : pressed
        rlc     l                   ;get next addr
        rlc     l                   ;last addr had no char
        ld      a,(hl)              ;get the key
        bit     2,a                 ;test for cntrl
        jr      z,ctrl
        rlc     l                   ;get next address
        ld      a,(hl)              ;read key
        bit     2,a
        jr      z,shift             ;exit if shift key pressed
        ld      a,0ffh              ;get no char code
        ld      (char),a
        xor     a
        ld      (flag),a            ;clear shift flag
        ret
;
ctrl:   ld      a,(flag)
        set     2,a
        jr      keyexit
;
shift:  ld      a,(flag)            ;get flag
        set     1,a                 ;set bit 1 flag
keyexit:ld      (flag),a
        ld      a,0ffh              ;make no char code
        ld      (char),a
same:   xor     a                   ;if char the same, return no char so as to
                                    ;remove the key repeat
        ret
;
minus:  ld      c,3                 ;set row count
        jr      found               ;exit
;
carret: ld      c,2                 ;set row count
        jr      found
;
```

```
colon:   ld      c,1             ;set row count
;
found:   ld      hl,txtble       ;point to lookup table
         ld      e,0             ;clear shift/control offset
         ld      a,(flag)
         bit     2,a             ;test for ctrl key hit last time
         jr      z,noctrl
         ld      e,48*2          ;make up control table offset
         jr      noshift
;
noctrl:  bit     1,a             ;test for the shift key
         jr      z,noshift
         ld      e,48            ;set shifted offset
noshift:ld       a,8             ;setup row count
         sub     c               ;calc offset
         ld      c,a             ;store in C
         ld      a,6             ;setup column count
         sub     b               ;calc offset
         ld      b,a             ;store in b
         call    cond            ;get table offset in A
         add     a,e             ;add shift offset
         ld      b,0             ;clear b
         ld      c,a             ;offset in C
         add     hl,bc           ;get character position
         ld      a,(hl)          ;get char in A
         ld      hl,char         ;get last char
         cp      (hl)            ;same as last ?
         jr      z,same
         ld      (hl),a          ;if not same, save the char
         ret
;
cond:    xor     a               ;clear a reg.
         cp      c               ;is c=0
         jr      z,hre           ;bypass if yes
mult:    add     a,6             ;inc by six
         dec     c               ;dec counter
         jr      nz,mult
hre:     add     a,b             ;add bit count
         ret
;
txtble:  defb    'TW EQRGS DAFBX CZV52 314N. , M'
         defb    '69-807YO'
         defb    0dh
         defb    'IPUHL:K;J'
;
;shifted characters
;
         defb    0               ;shift T
         defb    0               ;shift w
         defb    0
         defb    0               ;shift e
         defb    0               ;shift q
         defb    0               ;shift r
         defb    0               ;shift g
         defb    0               ;shift s
         defb    0               ;cntrl
         defb    0               ;shift d
         defb    0               ;shift a
         defb    0               ;shift f
         defb    0               ;shift b
         defb    101             ;shift x
         defb    0               ;null
         defb    0               ;shift c
         defw    0               ;null
         defb    '%" #!$'        ;shifted 52 314
         defb    0               ;shift n
```

```
		defb	'>'		;shift .
		defb	0
		defb	'<'		;shift ,
		defb	0		;null
		defb	92		;shift m
		defb	'&)=('		;shifted 69-8
		defb	'@'		;shift 0
		defb	0		;shift 7
		defb	0		;shift y
		defb	0		;shift o
		defb	0		;shift cr
		defb	0		;shift i
		defb	0		;shift p
		defb	0		;shift u
		defb	0		;shift h
		defb	'?'		;shift l
		defb	'*'		;shift :
		defb	'/'		;shift k
		defb	'+'		;shift ;
		defb	0		;shift j
;
;control characters
;
		defb	'T'-040h	;control t
		defb	'W'-040h	;control w
		defb	0
		defb	'E'-040h	;control e
		defb	'Q'-040h	;control q
		defb	'R'-040h	;control r
		defb	'G'-040h	;control g
		defb	'S'-040h	;control s
		defb	0
		defb	0		;control d
		defb	'A'-040h	;control a
		defb	'F'-040h	;control f
		defb	'B'-040h	;control b
		defb	'X'-040h	;control x
		defb	0
		defb	'C'-040h	;control c
		defb	'Z'-040h	;control z
		defb	'V'-040h	;control v
		defb	0
		defb	0
		defb	0
		defb	0
		defb	0
		defb	0
		defb	'N'-040h	;control n
		defb	0
		defb	0
		defb	0
		defb	0
		defb	cr		;control m
		defb	0
		defb	0
		defb	0
		defb	0
		defb	0
		defb	0
		defb	'Y'-040h	;control y
		defb	'O'-040h	;control o
		defb	0
		defb	ht		;control i
		defb	'P'-040h	;control p
		defb	'U'-040h	;control u
		defb	bs		;control h
```

```
            defb    ff                  ;control l
            defb    0
            defb    'K'-040h            ;control k
            defb    0
            defb    1f                  ;control j
;
endadr      equ     $
.xlist                                  ;xlist is so you don't see heaps of ff's
                                        ;in a printer listing !!
bytes       defl    2048-(endadr-begadr)    ;figure out how big the program is
            rept    bytes               ;and fill the rest of it with ff's
            defb    0ffh                ;means that the eprom programmer works
            endm                        ;heaps faster.
            .dephase
            end

C:\rbbs>
```

Another VZ200 RTTY System

Lloyd Butler VK5BR
18 Ottawa Avenue, Panorama, S.A. 5041.

Generation of RTTY tones and BAUD rate clock can be controlled from the keyboard using a programmable interval timer. Experimental hardware and associated computer programme have been developed incorporating such a system for RTTY on the VZ200.

Armed with no previous expeience in RTTY, the writer set out to adapt a VZ200 computer for the purpose. Had the ETI-Dick Smith kit been available at the time, the project might never have been started and purchase of a kit might have been the way to go. Notwithstanding this, the project was proceeded with, to an operational state, using a number of different ideas which could well be of interest to others experimenting with the VZ200.

THE HAREWARE

The circuit of additional hardware, plugged into the VZ200 memory expansion socket, is shown in figure 1. Serial encoding and decoding of the teletype signal is carried out by a communications interface (8251 USART). The teletype programme is stored in a 2732 4 K Byte EPROM.

An important difference, to that of the ETI system, is the inclusion of an 8253 interval timer which contains three independant programmable 16 bit counters. Two of these counters are used to generate the two teletype tones divided down from the computer clock. The third counter is used to feed the USART and determine the BAUD rate. The advantage of this system is that there are no oscillators to adjust for correct frequency and tones and BAUD rate are set to an accuracy, determined by crystal control in the computer. Furthermore, the tones and the BAUD rate are under the control of software and can be changed for the computer keyboard.

The USART BAUD rate control clock is fed at sixteen times the BAUD rate. (Note: Although one times the BAUD rate can be used, errors result in decoding if the BAUD rate is not exactly synchronous to that used on the signal being received.)

Output tones are square wave and these are shaped to reduce harmonics by an RC filter network.

THE PROGRAMME

The programme developed by the writer provides selection of the following modes of operation from the keyboard —

1 ASCII or BAUDOT codes

2 BAUD rates — 45.45, 50, 56.92, 74.2, 100, 110, 150, 300, and 600 Hz.

3 Tone pairs —

Mark-Hz	SPACE-Hz
1275	1445
1275	1700
1275	2125
2125	2295
2125	2550
2125	2975

4 Two buffer stores, 1000 Bytes each.

5 Message resident in programme.

CQ de VK5BR
RYRYRY...etc
The quick brown fox..................................etc 1234567890
de VK5BR Lloyd

6 Selection of split screen or normal screen. (Split screen is used to load the buffer at the same time as receiving. Normal screen allows full use of the screen for receive only).

7 Clear screen control.

8 Reverse receive BAUDOT letters/figures. (This is useful if a letter/figure switch character is lost or one is interpreted when it shouldn't be. Sometimes a whole line can be lost when this happens unless reverse is operated).

Included in the programme is automatic insertion of carriage return and line feed at the first space after each and every 50 characters. This is a good feature to prevent printers running over the end stop and over-riding the necessity to put in CRLF when required. Sending BAUDOT, letter/figure control is also initiated on the character after each space, independant of any control put in because of a letter/figure change. This reduces the error to one word in the event of a wrong change in decoding at the receive end.

The programme resides in an EPROM at memory locations COO3H to CDOAH. RAM space utilised in 8000H to 8900H. The RTTY programme is initiated from the basic monitor with two POKE statements and an X= USR (x). Return to basic monitor can be carried out at any time with simple commands from the keyboard.

The programme is written in instructions suitable for 8080/8085 or Z80 processors, but is dedicated to the VZ200 in that it calls in the resident VZ200 keyboard, character print and beep routines.

DECODING

From the point of view of reducing component parts, a phase locked loop system (such as the XR 2211 circuit) is the simplest way to go. On the other hand, all the experts say, that in the presence of noise, better performance is achieved with a filter type system and essential for reception on the HF bands.

Many circuits have been published for both types of decoders and since the decoder design has no bearing on the computer hardware and software design, further comment will be avoided on design. At this point it must be pointed out that it would be a fairly complex decoder which could cope with all the BAUD rates and tone combinations available for transmission from this computer system. These were selected from standards recommended in Amateur Radio last year, and were all included just in case they were required. It is unlikely that other than 45 or 50 BAUDS and 2kHz tones will get used on the experimental unit assembled and at present it is being operated with a 2kHz type filter system which will accept up to 100 BAUDS.

ASSEMBLY

The VZ200 attachment was made up using a general purpose printed circuit card, suitable socket fitted and hard wired. For the present, the attachment is unshielded and causes some interference to radio receivers. Fitting of a metal enclosure is a job still to be tackled. What is really needed is some industrious person to layout the printed circuit card and design an appropriate housing.

SUMMARY

A RTTY system for the VZ200 computer has been developed as an experimental exercise. Transmission tones and BAUD rate clock are generated from the computer clock. The programme is operational but no action has been taken to lay out an easily assembled printed circuit card and shielded enclosure.

The programme has not been included as it is 3338 Bytes of machine language. Those who contemplate construction many consult the writer about copying the programme. AR

I LIKE AMATEUR RADIO

I like amateur radio;
I really think it's fine
That I'll still be a "YL"
If I live to ninety-nine.

I like amateur radio,
And getting on the air,
Making friends around the world
And contacts everywhere.

You can talk to Lapps in Lapland,
Nepalese in Katmandu,
Malays in Kuala Lumpar,
Or Peruvians in Peru.

You can talk to dukes and dustmen,
Or communicate in Morse,
Experiment with A T V,
And RTTY of course.

Put together bits and pieces,
(Though at first the prospect balks;)
A diode here, condenser there,
And — listen to that — it talks;

Experiment with aerials,
It looks real good on paper;
But getting that lot in the air
Is quite another caper;

You can enter in a contest,
Gather points for an award,
Join a DX net, or "ragchew",
One thing's sure, you're never bored.

Yes, I like amateur radio,
And all the friendly sounds,
Removed from all the trouble and strife
With which this world abounds.

It's a satisfying hobby,
It will certainly do me;
Til they write beside my name the words
"Became a silent key." JOY COLLIS.VK2EBX

 1 of 2 p. 10-11.

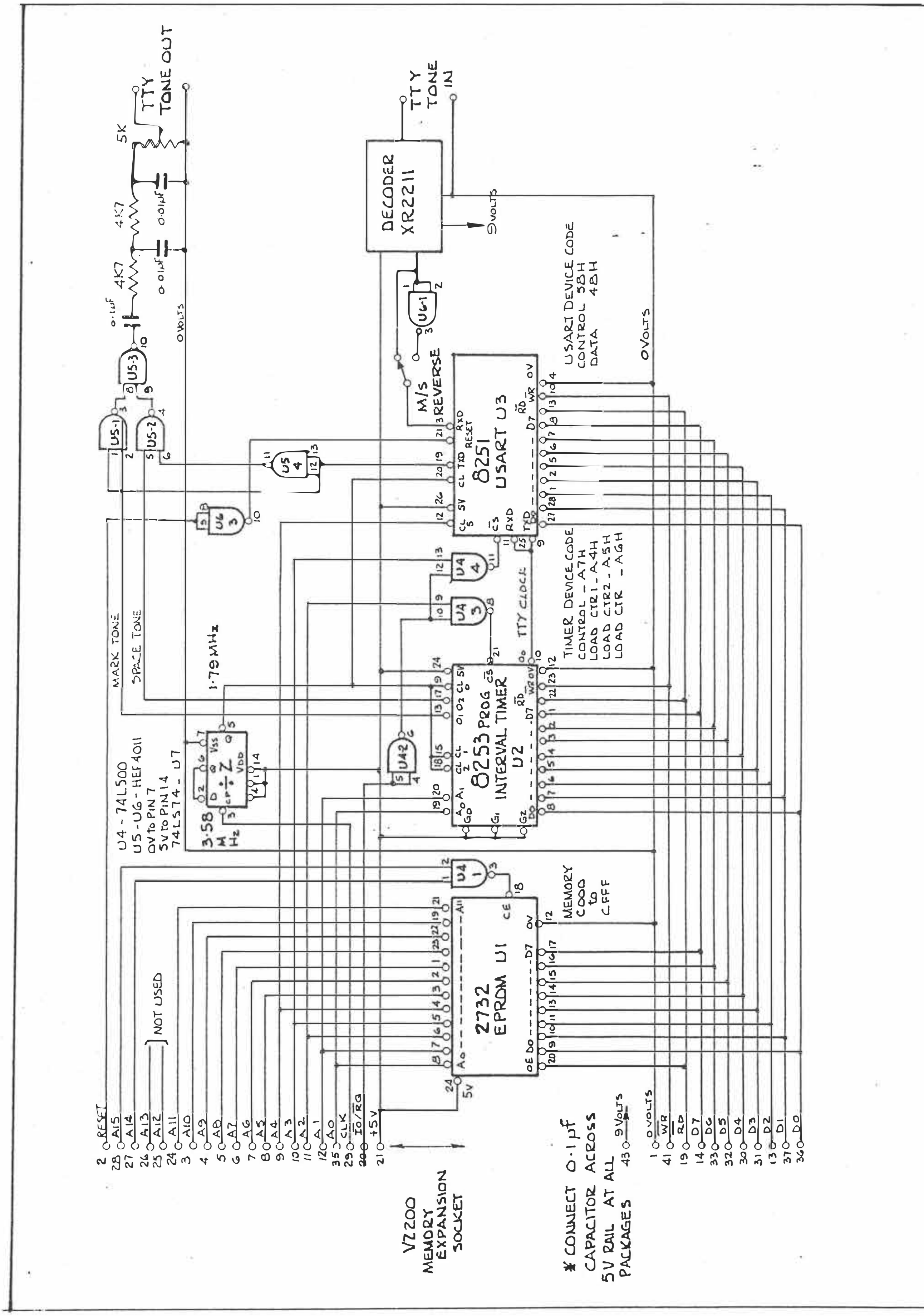


Figure 1 — VZ200 RTTY Attachment.

2 of 2.

MORSE CODE ON THE VZ200

A previous article described an adaptor to operate RTTY on the VZ200 computer. The adapter has now been modified to include Morse code.

Lloyd Butler VK5BR
18 Ottawa Avenue, Panorama, SA. 5041

Expansion of the programme resident in the EPROM and minor changes to the wiring, have expanded the VZ200 RTTY adaptor to include encoding and decoding of Morse code. Morse speed can be varied over a range of approximately five to 35 words per minute. Resident messages, buffer storage and split screen operation, all used on RTTY, are also available for Morse operation.

HARDWARE CHANGES

To interface for Morse code, the 8251 USART functions DSR and DTR are used as one bit input and output ports respectively. DSR is simply wired in parallel with the existing data input (RXD). DTR is wired via a spare gate (V6-2), which is used to key the tone output from gate (U5-3). The circuit changes are illustrated in Figure 1.

For Morse code, the output tone is set at 2125Hz by the software and this can be used to feed the speech input of a transmitter. In a single side-band transmitter, CW transmission (A1) is generated and on a transmitter where carrier is not suppressed, MCM transmission (A2 or F2) is generated. Of course the latter is only permissible above 52MHz.

MORSE FORMAT

Morse format is based on the following:

Dash = three dots length
Space between dot or dash elements = one dot length
Space between characters = three dots length
Space between words = seven dots length

Speed is controlled by a selection code of one to eight and for the two lowest speeds (below 10 WPM), the spacing is increased to the following:

Space between characters = five dots length
Space between words = 13 dots length

There are a number of special Morse characters which are not available on the keyboard and not available as printed characters. These have been equated to available characters as follows:

Error = asterisk (*)
Double dash = dash (—)
Wait = plus (+)
Start of message = less than (<)
End of message = equals (=)
End of work = at (@)

Error is transmitted as six dots, instead of the standard eight, because six elements per Morse character is the maximum the system can process.

Morse characters are generated from a look-up table, one byte per character. Bits two to seven are used to store the individual elements of a character, zero representing no element or a dot and one representing a dash. Elements are justified left, with the last element sent, always in bit seven. The numeric value formed by this is added to the number of elements in the character and the sum is the value stored in the look-up table. For up to five element characters, it is an easy matter to extract the number of elements from bits zero to two and the dots and dashes elements from bits three to seven. For six element characters, there is an overlap on bit 2 and summing causes bit carry on four of these (parenthesis, comma, colon, and semi-colon). To detect these is a bit tricky. The logic is to look for a one in either bits four or five and binary 010 in bits zero to two. If this logic is satisfied, the number of elements is assumed to be six and six is subtracted from the byte value to obtain the element format in bits two to seven.

Some examples of look-up table coding are shown in Figure 2.

OPERATION

Morse can be sent on line, direct from the keyboard and characters are encoded at the selected speed by the software. In this method of operation, character and word spacing are

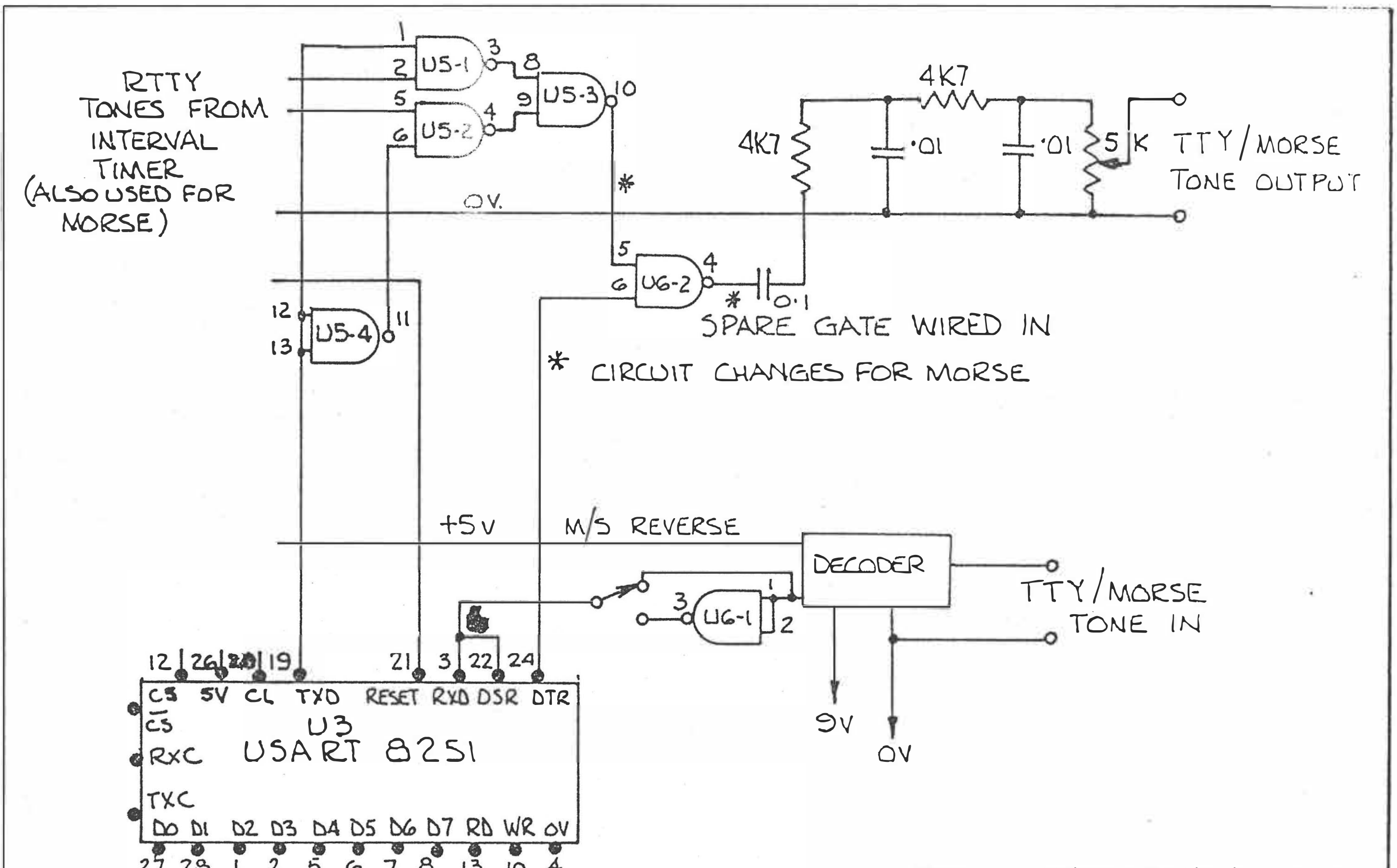


Figure 1 — Additions to the Circuit for Morse.

p. 19-20 1 of 2.

Figure 2 — Examples of Table Coding for Morse.

MORSE CODE	BINARY VALUE (BIT No)	HEX VALUE
	7 6 5 4 3 2 1 0	
Letter B — . . .	0 0 0 1 0 1 0 0	14
	code 4 elements	
Interrogation (?) . . — — . .	0 0 1 1 0 0 0 0	30
	code + 1 1 0	6
	6 elements = 0 0 1 1 0 1 1 0	36
Comma (,) — — . . — —	1 1 0 0 1 1 0 0	C C
	code + 1 1 0	6
	6 elements = 1 1 0 1 0 0 1 0	D2
	Carry of Bit 2 into Bits 3 & 4	

determined by the time taken to move from one key to the next and, it seems to the writer, that a lot of practice would be needed to control the spacing correctly.

Morse is better sent by releasing the message from a pre-loaded buffer so that character and word spacing is accurately controlled by the computer. Using this method of operation, when communicating with another station, it is necessary to load the buffer at the same time as the other station is being received. This is common practice with RTTY operators using computers with split screen displays.

For RTTY, characters are encoded and decoded by the 8251 USART and the device is addressed by the computer for a very small proportion of the time. The rest of the time is available for other purposes including accessing the keyboard and loading the buffer, hence there is no problem in preparing the signal for transmission whilst the received signal is being decoded.

For Morse code, characters are encoded and decoded by timing loops called in by the main programme routine and while this is going on, access to the keyboard to load the buffer is denied. The obvious answer to the problem is to access the keyboard via an interrupt, however to make things difficult, the Z80 interrupt is already used by the VZ200 operating system. This calls an interrupt every 20 milli-seconds on video vertical retrace.

Steve Onley described a method to make use of this 20 milli-second interrupt in Electronics Today International (ETI), May 1985. Your own interrupt is placed in series with that of the operating system so that it too can interrupt the main programme loop every 20 milli-seconds. The method described has been adopted for accessing the keyboard and loading the buffer in Morse operation.

Owing to peculiarities of the VZ200 system, keyboard access using this interrupt inhibits repetitive generation of a character, that is, you have to press the key each time a character is to be generated. This is not such a bad thing as it stops generation of more than one character if the key is accidentally pressed too long. The reason for the peculiarity is not clear as we do not have access to information on the VZ200 operating system.

The interrupt system works very well for loading the buffer, but a problem was found in attempting to generate Morse characters this way in real time. Because of the peculiarity discussed, a key pressed too soon, before the previous character is finished being transmitted, fails to generate a character and locks in this condition until the key is released and pressed again at the end of the previous character. Because of this problem, the interrupt is only used for loading the buffer and in all other modes of operation, the keyboard is accessed from the main programme loop. Using this method of access, the key can be kept pressed and the new character is sent, following a three dot length space, at the end of the previous character.

MEMORY

The combined RTTY and Morse programme package fully fills the 4k byte EPROM. A certain amount of programme trimming and re-arrangement had to be carried out to fit it in. The programme is loaded in memory COO3H to CFF9H. RAM space used is 8000H to 8900H.

Based on information given by Jim Rowe in ETI, July 1985, the memory allocation should be suitable for both the VZ300 and VZ200 computers. A VZ300 has not been available to check it out, but the adaptor is expected to also work on the VZ300. There appears to be a change in clock frequency in the VZ300 from 3.580 to 3.540MHz. This will cause a shift in Baud rate and tone frequencies, but insufficient to be of significance.

CONCLUSION

The unit works very well on both RTTY and Morse code. The Morse decodes over a wide tolerance in reference to the speed selected. The writer was surprised how well it manages to decode hand sent Morse in which timing is not precisely defined. Noise interference is reduced by feeding the input signal via the RTTY decoder filters, but it does not perform as well as the human ear in separating Morse from noise. No doubt this could be improved if frequency shift keying were used.

Morse sent from the buffer sounds copper-plate, as one would expect fully controlled by the computer. On line from the keyboard, the writer found it difficult to maintain constant character spacing, but this is probably a matter of practice on the keyboard. AR

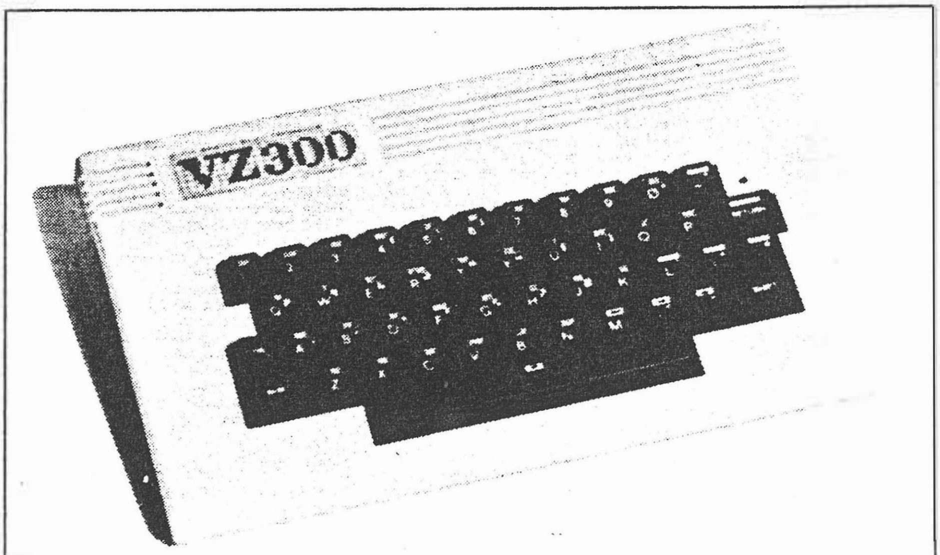
MODIFYING THE VZ200 16K EXPANSION MODULE FOR THE VZ300

Steve Olney

This article describes a method of remapping a DSE VZ200 16K RAM expansion module preventing overlap of memory space when used on a VZ300. The cost is limited to the price of one integrated circuit chip plus a single-pole double-throw switch if dual VZ200/300 compatibility is desired. The modification is fitted inside the expansion module case.

MANY OF YOU who have updated to the new version VZ300 must be disappointed to realise that although the VZ300 comes with much more internal RAM as standard (18K as against 8K for the VZ200), use of your old VZ200 16K expansion module on the VZ300 only results in the same total memory as that which was available on the older VZ200 with the expansion module plugged in.

The reason for this becomes clear when a comparison is made between the memory maps of the VZ200 and the VZ300 as shown in Figure 1. If a VZ200 16K expansion module is plugged into a VZ300, about 10K of the expansion RAM overlaps memory space already provided to the VZ300 internally. This results in only 6144 bytes of extra memory. In order to make proper use of the expansion memory space, the start of the VZ200 expansion module needs to be moved or remapped to the end of the VZ300 internal memory instead of somewhere in the middle. For more details on the memory map of the VZ200 and VZ300, refer to Jim Rowe's informative article on the VZ300, ETI July 1985.



The object of this article is to provide information sufficient to modify a VZ200 16K expansion module to be used on both your VZ200 as well as your new VZ300.

Before proceeding there are a few words of advice for those wishing to undertake the modification:

1. Because you are modifying an existing working unit, this project is intended for those with reasonable soldering skills and at least some experience with digital components. If you are unsure, enlist the aid of someone capable (and willing) to carry out the modification.

2. Remember, modification to your module will render the module warranty void, although I expect most modules would be out of warranty anyway.

3. The modification details provided are for printed circuit boards identified by the '700352 F' designation. If you find a different number near where the seven ICs are located, then be careful to ensure that all mechanical details supplied here agree with your board. If they don't, I advise you not to proceed unless you have sufficient knowledge to adapt the circuit for that board.

The circuit

Modifying the address decoding logic to remap the expansion RAM only requires two extra AND gates, so half a 74LS08 IC is all that is really needed, but I used NAND gates. The reason for this is that quite often, when a design is completed, extra input sig-

1 of 3.

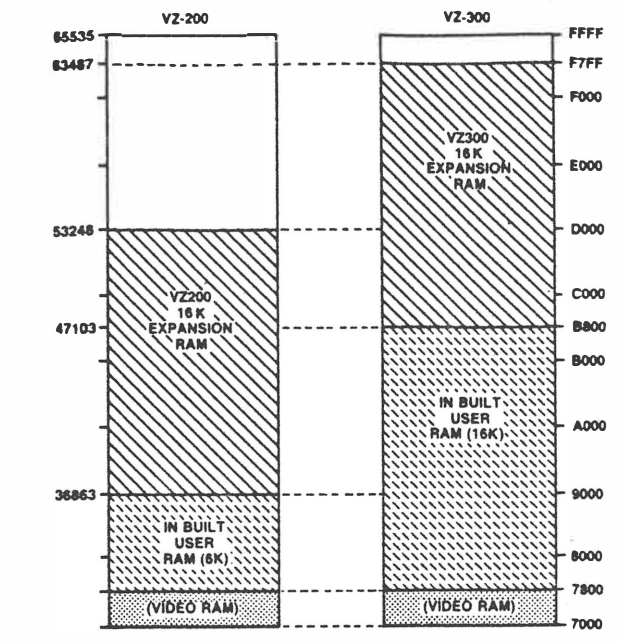


Figure 1. The memory maps for the VZ-200 and VZ-300. Note how their internal and expansion RAMs cover different address ranges.

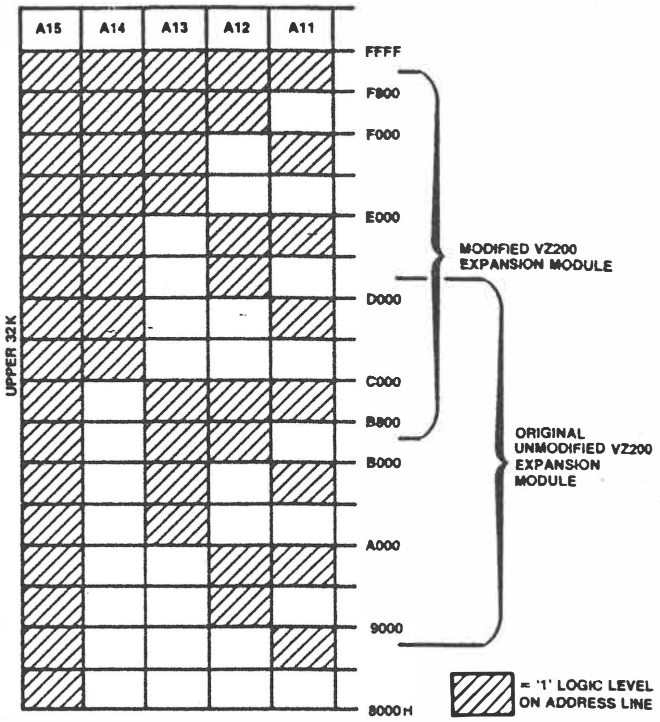


Figure 2. Address decoding map.

nals or controls are required. Because NAND or NOR gates can be configured to implement all of the basic logic functions, they are often used in at least some part of a circuit — even when that part could be more efficiently designed with other logic units. This is done with the view that if modification is required, then spare NAND or NOR gates allow some flexibility.

To further illustrate this point, it occurred to me, after working out the circuit, that it might be useful to have a block of RAM separated completely from the contiguous internal RAM for such purposes as having a reserved area of memory for running machine code programs, or implementing a printer buffer in RAM under software control. To do this the 16K RAM pack could be remapped to extend from C000H to the top of addressable memory, FFFFH. This would result in a 2K byte gap (for the VZ300 only) between the end of internal memory and the start of the expansion memory. When the BASIC interpreter seeks the top of memory, it is unable to jump this gap and so the top of memory pointers are set to the end of internal memory. This creates a reserved 16K block of RAM from C000H to FFFFH. That is, the top of memory pointers in BASIC are set to the same values as for a VZ without expansion module. This would still mean, of course, 18K for the VZ300, but only 8K for the VZ200. If the original circuit was implemented with AND gates the circuit would have to be re-designed. However, because NAND gates are being used, one of the paralleled inputs of one NAND gate can simply be switched to implement this change. This is shown in Figure 3.

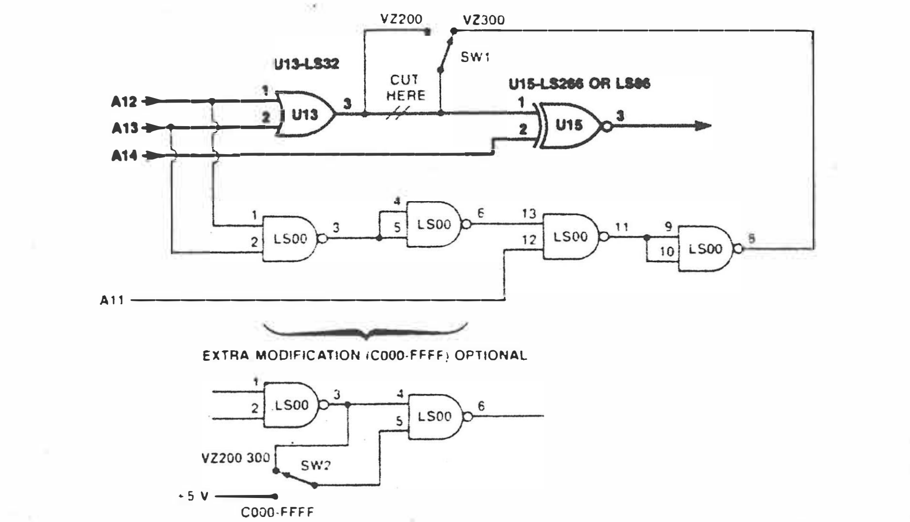


Figure 3. Modification details.

The decoding logic

Those who are not curious about the decoding logic details can skip this section and go straight on to the modifications.

To work out the new decoding required, a graphical method was used. By looking at Figure 2, we can see that A14 = 1 (address line 14 = 1) covers from C000H to FFFFH (49152 to 65535 decimal). However, this is 2K bytes too high; the top 2K bytes need to be disabled, and 2K bytes added to the bottom, in order to enable a block extending from B800H to F7FFH (47104 to 63487 decimal). That is, from the end of the VZ300 internal memory on up. It might be noted that from F800H to FFFFH (where the memory should *not* be enabled) A13, A12 and A11 are = 1. Also from B800H to C000H (where the memory *should* be enabled) A13, A12 and A11 are again = 1. The only difference is that A14 = 1 in the first case, and = 0 in the second case. In other words, the memory should be enabled when A14 = 1 or when A13 and A12 and A11 all = 1, except when they all (A14-A11) = '1' at the same time. In logical shorthand this is written as:

A14 ⊕ (A13•A12•A11)

2 of 3.

where '⊕' is the sign for the logical XOR function, and '•' is the sign for the logical AND function.

Looking at the original circuit (Figure 3) it can be seen that the XOR function is available with A14 already connected (pin 2 U15), so if the A13•A12•A11 signal is connected to the other input (U15 pin 1) then the required memory enable signal is available on U15 pin 3. The required input is supplied to U15 pin 1 by the four NAND gates of the added 74LS00 IC.

Modification steps

The following steps are the hardware modifications that need to be carried out to effect the change to the expansion module.

Turn over the module to find a sticker with the number 8 on it. This (apparently) indicates that the module is configured to expand on 8K VZ200.

Remove the six screws from the bottom of the case and gently separate the top of the case by means of a flat bladed screwdriver. Do this at the connector end of the module first, as there is a tendency for the cover to jam if it is pulled off at an angle. This will reveal a pc board to which a metal shield is attached by six soldered tabs.

Use solder wick or a solder sucker to remove as much solder as possible from the six tabs holding the metal shield in place, *gently* freeing the tabs from the board one at a time. Remove the metal shield.

At this point the component side of the board is visible with the physical layout as shown in Figure 4. Check to see that the board is marked with the 700532F designation. Hold the board with the component side towards you and the seven ICs at the top, and the discrete components (diodes, transistors etc) at the bottom (as in Figure 4). The middle IC of the seven ICs should be a 74LS86 or a 74LS266. In either case the modifications are the same.

Find the track on the component side of the board which runs from pin 3 on the 74LS232 to between pins 12 and 13 on the 74LS86/266 and cut it carefully with a sharp knife as in Figure 5.

Decide where to mount the SPDT change-over switch. I soldered a right angle pcb mounting type to the board itself (see Figure 4). You will probably need to shorten the terminal legs of the switch first and make sure the switch will not foul the metal shield when it is re-fitted. Another arrangement would be to mount the switch through a hole drilled in the top part of the plastic case. This is satisfactory providing the switch protruding out does not foul the printer or joystick interface plugged in next to it.

Using multi-strand insulated wire (wire stripped from rainbow ribbon cable is excellent) connect the centre (or common) terminal of the change-over switch to pin 1 of the 74LS86/266, then connect pin 3 of the 74LS32 to one side of the switch (this position will select normal VZ200 operation).

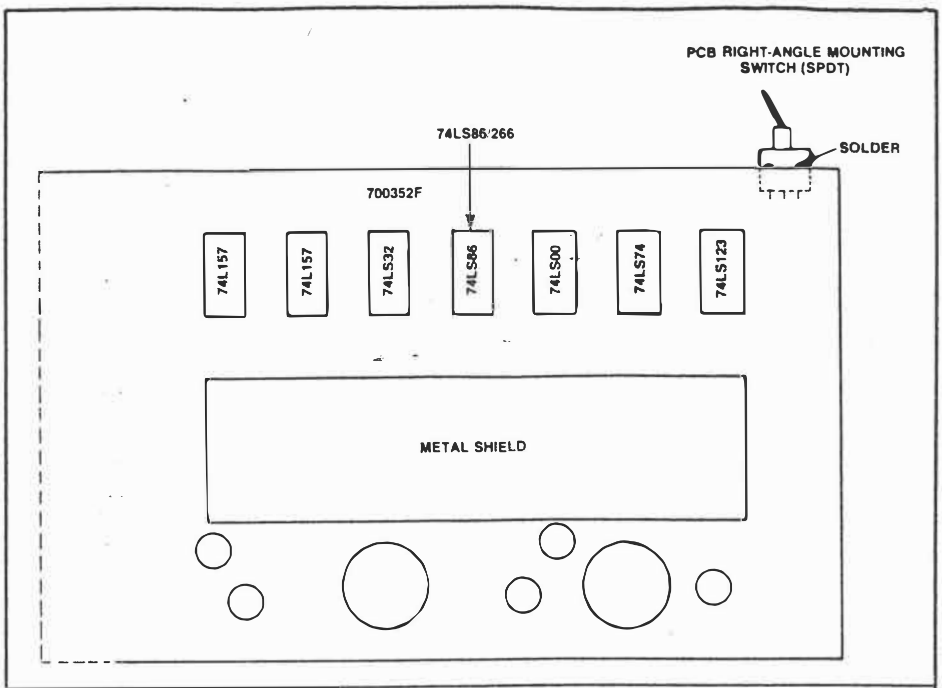


Figure 4. Component-side view.

Carefully bend all the pins *except* pins 1,2,7,12 and 14 on the 74LS00 at right angles to their original positions and carefully solder 'piggy-back' style pins 1,2,7,12 and 14 of the 74LS00 to pins 1,2,7,12 and 14, respectively, of the 74LS32.

Join pins 3, 4 and 5 on the 74LS00 together and solder them. Also join pins 9,10 and 11 together and solder. Join pin 6 to pin 13 using flexible multi-strand wire, then join pin 8 of the 74LS00 to the remaining side of the switch to give the VZ300 mode.

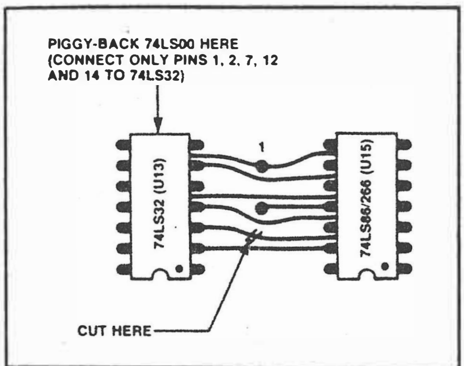


Figure 5. A closer view of the component side and directions.

Testing

That completes the hardware modification and the module is now ready for testing in your VZ300. Go over the modification carefully, making sure the wiring is correct and look out for solder bridges. With the power off, plug in the modified module, switch to VZ200 mode, and turn on the power to the computer. Type in the following

```
PRINT PEEK(30897) +
256*PEEK(30898) <RETURN>
```

If everything is OK, the response should be

```
53247
```

Now switch off the power to the computer, switch to the VZ300 mode and then switch the power back on. Type in the above line again. This time the response should be

```
63487
```

If any of the above two responses are not obtained, then switch off immediately, and re-check the modification looking for wiring mistakes or solder bridges.

By comparing these two responses with the response obtained without an expansion module plugged into the VZ300, it can be seen that the modification enables all 16K (16384) bytes of the expansion memory instead of only 6K (6166) bytes of the standard VZ200 module. That is:

- top of memory VZ300 alone = 47103;
- top of memory VZ300 + unmodified module = 53247 (6144 bytes extra);
- top of memory VZ300 + modified module = 63487 (16384 bytes extra).

Extra modifications

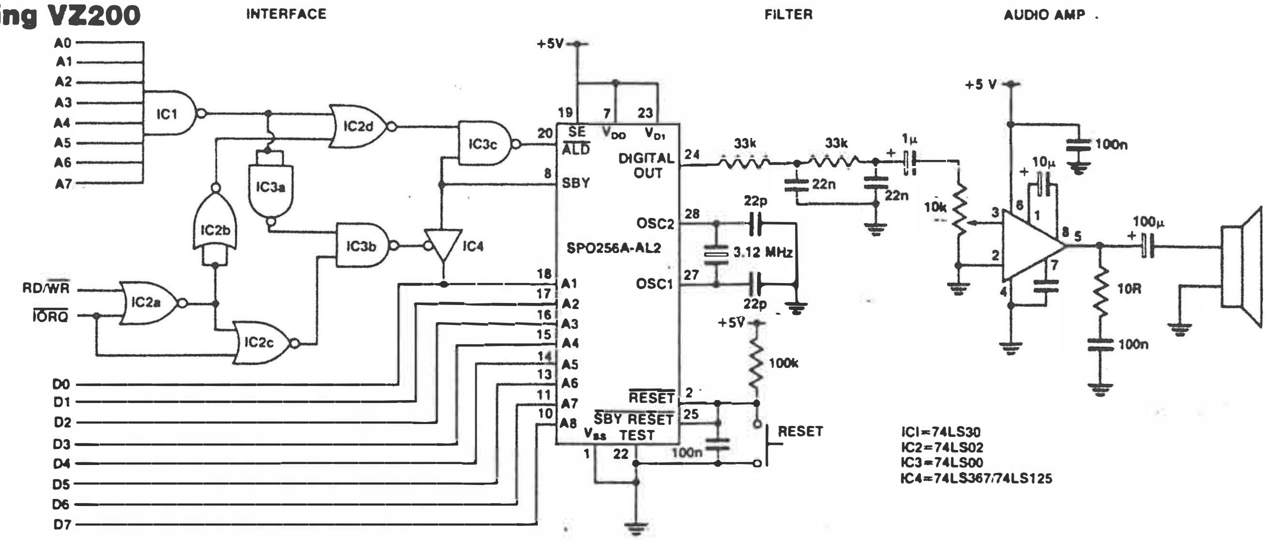
Before the module is re-assembled, an extra modification can be made, as mentioned earlier. This is to remap the expansion module to the top of addressable memory for reasons outlined before. This involves adding an extra change-over switch as shown in Figure 3.

Note that any of the switch connection positions can be replaced by direct wiring if operation in that mode is permanently required.

Happy Hacking!!! ●

3 of 3.

Talking VZ200



Matthew Bennets of **Corowa NSW,** sent us this circuit designed to allow speech processing from the VZ200 computer. You could adapt the idea to any computer where you can control seven address and seven data lines, plus two control lines.

Centrepiece of the circuit is the Radio Shack SPO 256A AL2 monolithic speech processor. An application manual is supplied with the chip which will give full details of addressing the words available on the chip. Mr Bennets has provided interface circuitry such that an address of FFH is required on the port, with the R/W and IORQ both logic '0' in order to access it. SDY (active low) is only active when speech is being output from pin 24. To input data into the chip the ALD pin must be active.

Output from the chip is taken from pin 24 to a low pass filter that removes the high frequency components from the output. It is then sent to an amplifier built around an op-amp.

SUPER II VZ200 MODIFICATION

Matthew Sorell

The VZ200 computer was one of the earliest of the really cheap, low performance computers on the market and as such it gained a loyal following. Over time, however, it's started to look a little too down-market. Its memory is ridiculously small, its keyboard is horrible and it lacks a number of features other computer users take for granted.

THERE ARE TWO solutions to this problem. One is to throw it away and buy a Microbee. The other is to be a bit more adventurous and see what can be done with the old carcass. In this project we show you how to increase the memory, fit a new keyboard, make it run faster, upgrade the power supply and provide a reset facility. You may carry out any or all of these modifications, as time, experience and inclination allow. It's called the Super II, for want of a better name.

Keyboard

The keyboard used in the prototype was a Digitran Golden Touch keyboard from Dick Smith. It was dirt cheap, as the keys had been coated with solder mask. Having cleaned and tinned the connections, it was as good as new. Another suitable keyboard is the Microbee.

If a numeric keypad is available, then this too can be connected, by means of the extra switches in parallel with the ones on the keyboard. On the prototype, the keypad included the digits 0 to 9, a period (.) and a RETURN key. If switches are not marked correctly, (eg, with graphic symbols), then these can be simply re-marked.

To start, remove any interface board from the keyboard. If the keys are a part of this board simply disable the interface circuit by cutting any tracks to it. By cutting tracks (on a pcb-based keyboard) and linking keys together, arrange the keys to form the matrix shown in Figure 1. With a keyboard using separate keys (eg, the Microbee), simply use wires to hook the keys together.

Extra keys, such as SHIFT-LOCK, LINE FEED or ESCAPE, should be left unconnected or removed if possible. On the prototype, only 58 of the 101 keys were used, the rest were removed and the holes covered with black insulation tape.

Additional keys such as a numeric keypad, can be wired in parallel with the key they correspond to. Shift keys should be wired in parallel.

Most keys will not be correctly marked for the VZ200. If this is the case, use Liquid Paper or similar to cover over the incorrect mark, and also over blank areas where a marking is required. Mark the key required by using a black pen or thin permanent texta.

If the keyboard is for the VZ200, then invert the colours or the graphics symbols on the keys (ie, black to white or vice versa). On the VZ300, the symbols have been corrected to the BASIC ROM. The colours can be marked on the keys 1 to 8 using the appropriate coloured permanent marker.

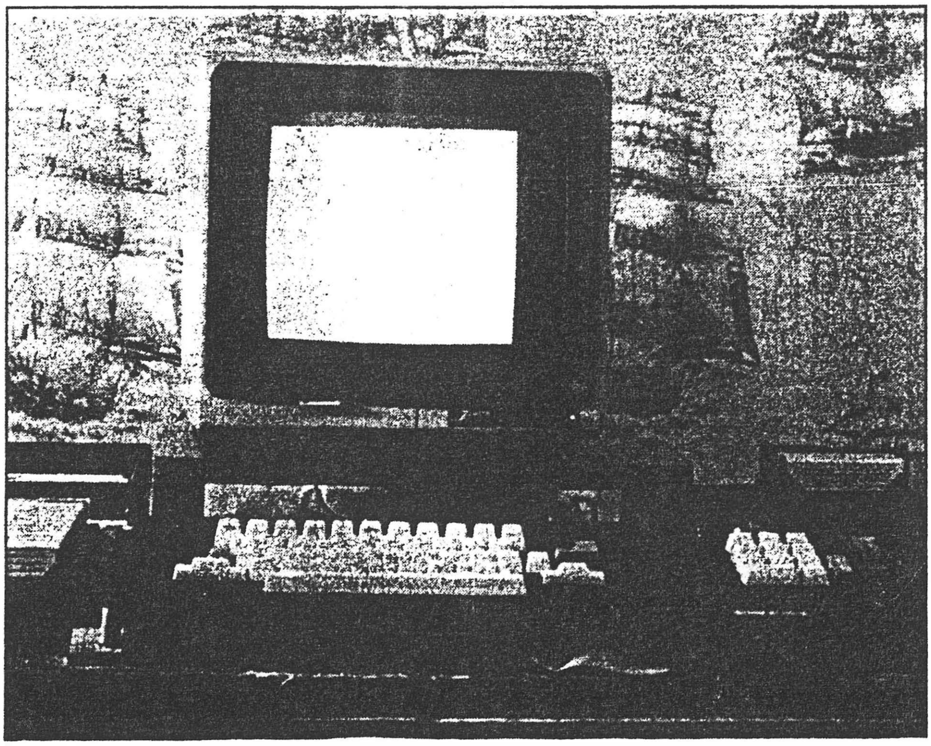
Control keys can be marked. On the prototype, only the control words for keys 1 to 8 were marked (ie, CSAVE/ CLOAD/ CRUN/ VERIFY/ LIST/ RUN/ END/ NEW). In addition, cursor control arrows, INSERT, RUBOUT, BREAK and INVERSE should be marked. Other keys may be marked, depending on your requirements.

When all the marking is complete, gently wipe each key with clear nail-polish. This protects the marking from being rubbed off, and provides a nice, silky finish to each key, if it is applied correctly!

Now that the keyboard is to your satisfaction, decide on the connector to be used. The prototype used a 16-pin DIP plug and a 16-pin IC socket. This is reasonably flat, and so can be mounted on the underside of the computer, but the IC socket is extremely hard to keep secure. Alternatively, a 15-pin D-connector can be mounted on the top of the computer (making sure it will fit when the lid is closed!), without the power supply connections.

Open the computer by removing the six back screws. Remove the four screws holding down the main pcb. Locate the 16 connections to the keyboard. Solder a wire on the track side of the pcb to each pad (see Figure 2). If +5V is required rather than the LED power signal, this can be obtained nearby. If no power indicator is required, the last two pads can be left without the extra wires.

Wire these new wires to the connector as shown in Figure 2. If a 16-pin DIP socket is used, cut a hole to suit in the bottom of the case, and use whatever you can to keep it there (Araldite, silicone rubber, plastic cement, etc). A 15-pin D connector can be mounted on the top of the case behind the "200" in the insignia. Make sure that the wire used is long enough, so that the case can be closed easily. In order to minimise wear, silicone rubber was smeared over the connections.



1 of 6.

PARTS LIST — ETI-687

MEMORY BOARD

Capacitors:
- C1 .... 47µ, 10V electrolytic
- C2-6 .... 100n ceramic or similar

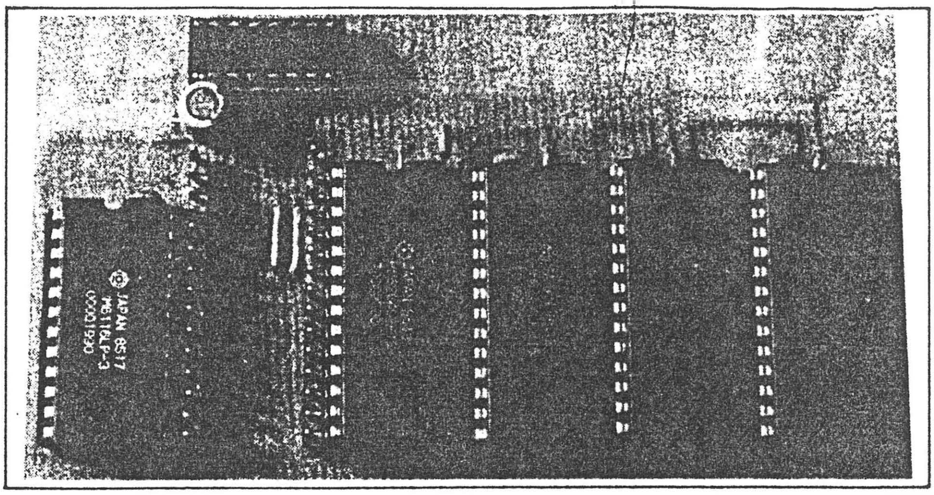
Semiconductors
- IC1 .... 74LS138
- IC2 6116
- IC3-6 6264

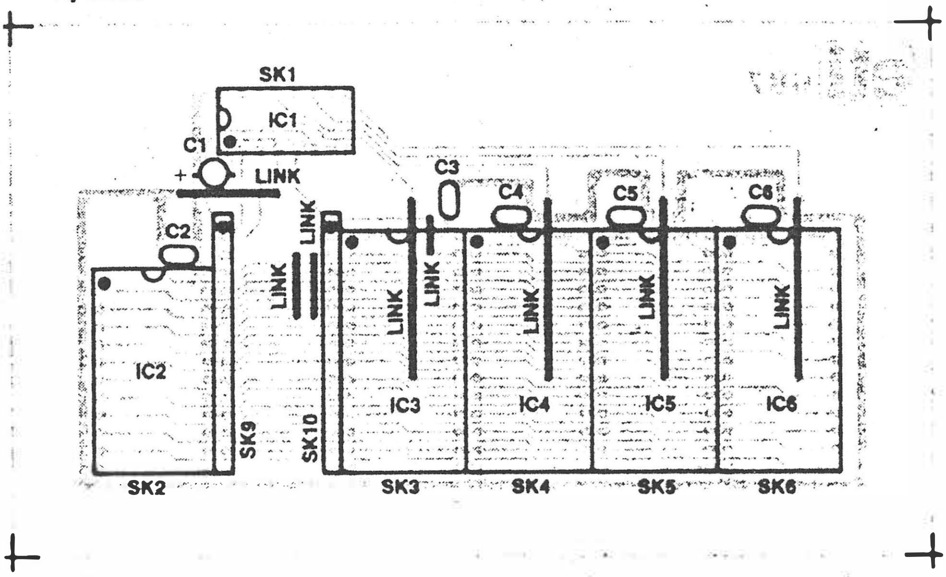
Plugs and sockets:
- SK1 .... 16-pin IC socket (for IC1)
- SK2 .... 24-pin IC socket (for IC2)
- SK3-6 .... 28-pin IC sockets (for IC3-6)
- SK7,8 .... 15-pin Utilux socket (for main pcb)
- SK9-12 .... 15-pin Utilux plugs (for both RAM boards)

Miscellaneous:
ETI-687 pc board; wire.

Price estimate: $60

KEYBOARD
Keyboard to suit: see text;
Socket and plug top suit: see text;
Wire;
Silicone rubber (optional);
Liquid Paper;
Black pen or thin permanent texta;
Coloured textas — green, yellow, blue, red, buff, cyan, magenta and orange;
Clear nail polish.


Memory board.



Make up a cable from the keyboard, connecting the signals as shown in Figure 1 (ie, D0-D5 and A0-A7) to a suitable 15- or 16-pin connector. An LED may be connected across the power and 0V signals, if a 16-pin connector is used and these signals have been wired in place internally. The power signal, if replaced with +5V, may be used as desired.

Check your wiring. When everything appears to be correct, reassemble the computer, plug in the power and the video plugs *only*, and turn on. If the computer gives the correct sign on message, then all is well. Check that all the keys on the normal keyboard function correctly, then plug in the new keyboard.

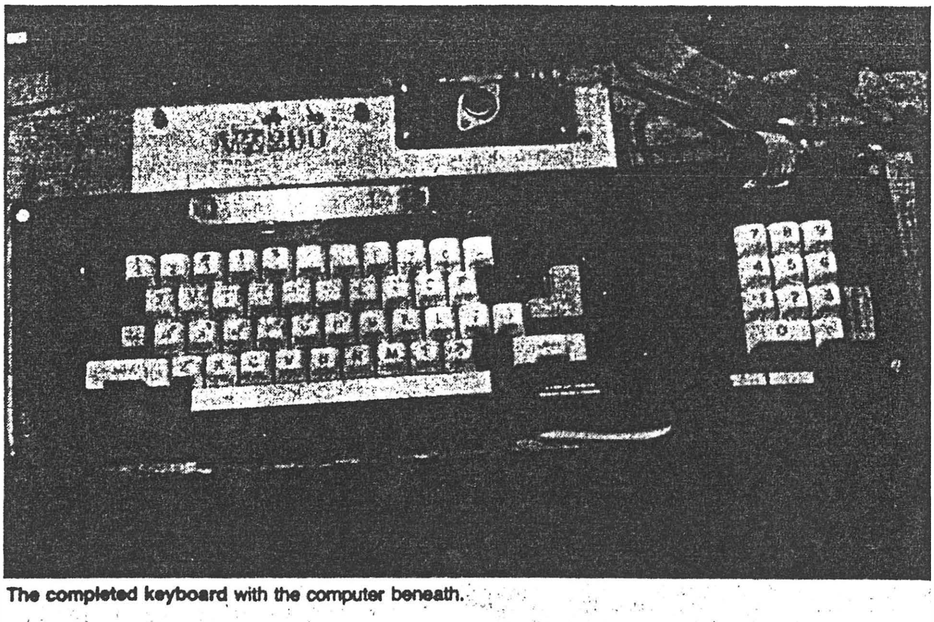
If everything has been wired properly, then the new keyboard should work. If the machine crashed when power was applied, reopen the box and look for both short and open circuits on the pcb. A multimeter is handy here. If the keyboard does not work, check your wiring.

You now have a keyboard to your satisfaction!

Memory

The new memory board replaces the standard 6K of the VZ200 with 34K of static RAM.

The design is relatively simple, using only five random access memories and an address decoder IC. This is because of the use of high density static RAMs which require almost no interfacing (unlike dynamic RAM, which requires multiplexed address



The completed keyboard with the computer beneath.

2 of 6.

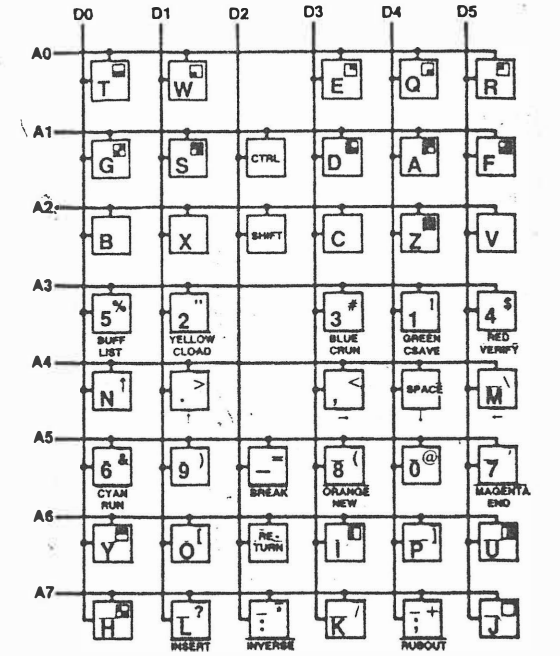


Figure 1. Keyboard matrix.

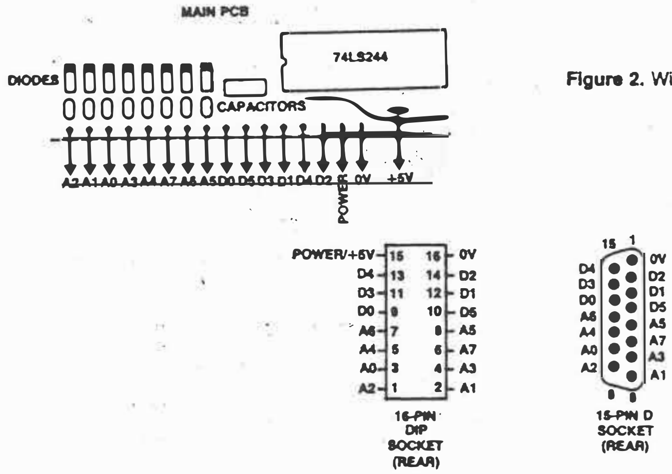


Figure 2. Wiring diagram.

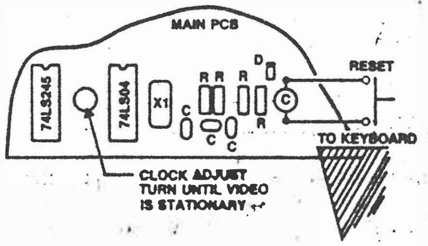


Figure 3. 74LS04 on VZ200.

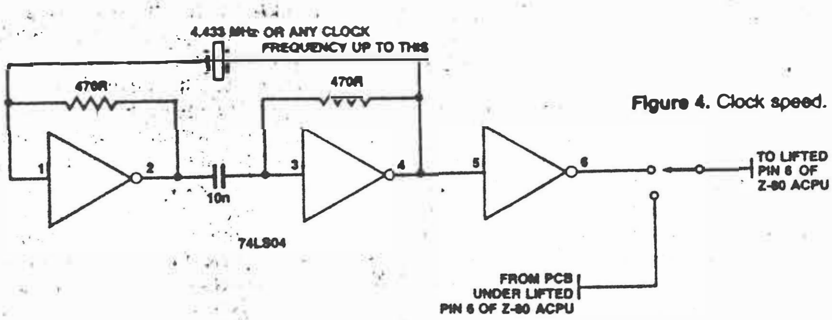


Figure 4. Clock speed.

lines plus refreshing) and are thus extremely easy to use. Two types are used: one 6116 2K RAM and four 6264 8K RAMs.

The pcb is single-sided and uses links in order to lower the cost. The board plugs in where the original board was fitted. Unfortunately, as sockets are used (the cost of the ICs makes this necessary), there is no room for the rf shield, however, I have found that there is no perceptible difference in the noise levels radiated by both shielded and unshielded computers.

34K is the maximum addressable RAM space in the VZ200 memory map. The expanded map looks like this:

0000-1FFFF: BASIX ROM 0
2000-3FFF: BASIC ROM 1
4000-67FF: reserved for ROM (eg DOS/RTTY etc)
6800-6FFF: input/output latch
7000-77FF: video RAM
*7800-7FFF: 2K user RAM (6116)
*8000-9FFF: 8K user RAM (6264)
*A000-BFFF: 8K user RAM (6264)
*C000-DFFF: 8K user RAM (6264)
*E000-FFFF: 8K user RAM (6264)

All peripherals are compatible with this set-up (except, of course, for memory expansion modules).

To begin check that the pcb has been drilled correctly and that there are no short or open circuits (a lens is handy). Solder in the eight links first. It is a good idea to use single-strand insulated wire, as some wires come very close to other contacts.

Now solder in the two 15-pin Utilux plugs. These require 1mm holes instead of the 0.8mm holes elsewhere. Then solder in the six IC sockets which are crammed together rather tightly in order to reduce board size, so be careful. The bypass capacitors should now be inserted. Take care with C1, the electrolytic. Check the board and put it aside (no ICs yet!).

Attack the VZ200. Remove the six back screws, lift the lid carefully (the keyboard will still be connected) and remove the four screws holding down the pcb. Desolder the main switch and the speaker (note the wires), and desolder the four lugs of the rf shield from the earth tracks. On the top, remove any braid to the rf shield then remove the shield. Behold! The RAM board is visible. Cut the short cables leading to the RAM board from the main pcb, remove the insulation and desolder each wire.

The contacts must be cleaned so that the connector can be inserted. I used a solder sucker, desoldering braid and a needle to clear the holes. *Be careful!* Overheating does wonders to the main pcb. If you lift any track, put in a link to replace it.

Insert the two 15-pin Utilux sockets to the main pcb. Check for lifted tracks. At the moment the computer will not work. In case anything goes wrong, it is a good idea to fit ▶

3 of 6.

HOW IT WORKS — ETI-687

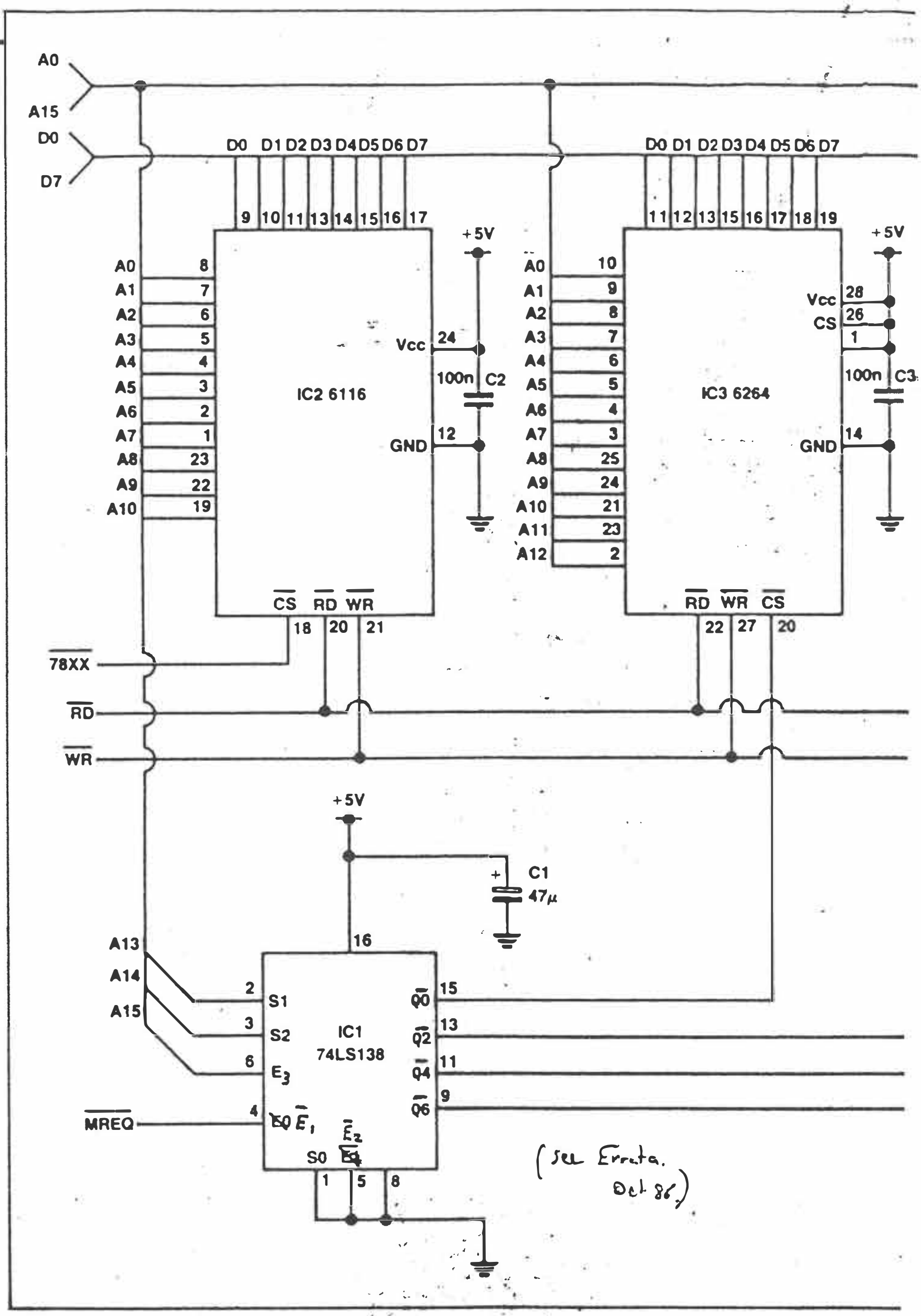
MEMORY BOARD
The RAM ICs require a simple TTL power supply. The data pins are connected directly to the data bus, as are the address lines. WRITE ENABLE is connected to the Z80 $\overline{WR}$ signal, and the OUTPUT ENABLE is connected to the $\overline{RD}$ signal. The 6116 is selected by an address decoder on the main pcb. The 6264s are selected through the 74LS138.

When $\overline{MREQ}$ is low and A15 is high, the 74LS138 will decode one of four combinations, depending on the status of A13 and A14. These signals are sent to the 6264s to enable the correct IC.

When any of the ICs are enabled ($\overline{CS}$ goes low), and a $\overline{WR}$ or $\overline{RD}$ signal is valid, the data bus will go out of tristate mode and either read data from the Z80 data bus into the address indicated on the address lines, or write data from the address on the address bus to the data bus. No other interfacing is required.

The decoupling capacitors are required for the stability of the power supply, as inductance occurs on the pcb with the power tracks.

KEYBOARD
The keyboard is continually scanned for any keys pressed. All address lines (A0-A7) are held low while the data bits (D0-D5) are checked. If any are found to be low then the computer knows that a key has been pressed and checks each address by holding the single address line low, and checking the data bits. SHIFT and CONTROL keys are also checked, and are used to derive the final character code.



(see Errata. Oct 86)

Utilux connectors to the 6K RAM board for testing. To do this the holes must be widened to 1mm. If you do this, now is a good time to check that the computer still works by plugging in the 6K board, reconnecting the power and the speaker and turning on. If the normal message appears, then proceed, otherwise check the main pcb for short and open circuits (a lens and/or a multimeter is handy) and good luck!

If all is well, insert the ICs into the new RAM board making sure they are inserted correctly then plug the board into the socket, making sure that all pins are in the right place. The board will not fit flatly into the socket because two RAMs get in the way, however the contacts are satisfactory.

Apply power. If the message appears (it will take longer than normal), then all is well. If not, check everything. Once again, good luck!

Once everything is working, screw down the main pcb and replace the back of the box. Test once more. Your VZ200 has 36K of RAM (including video RAM) . . . more than most home computers on the market!

Some other modifications

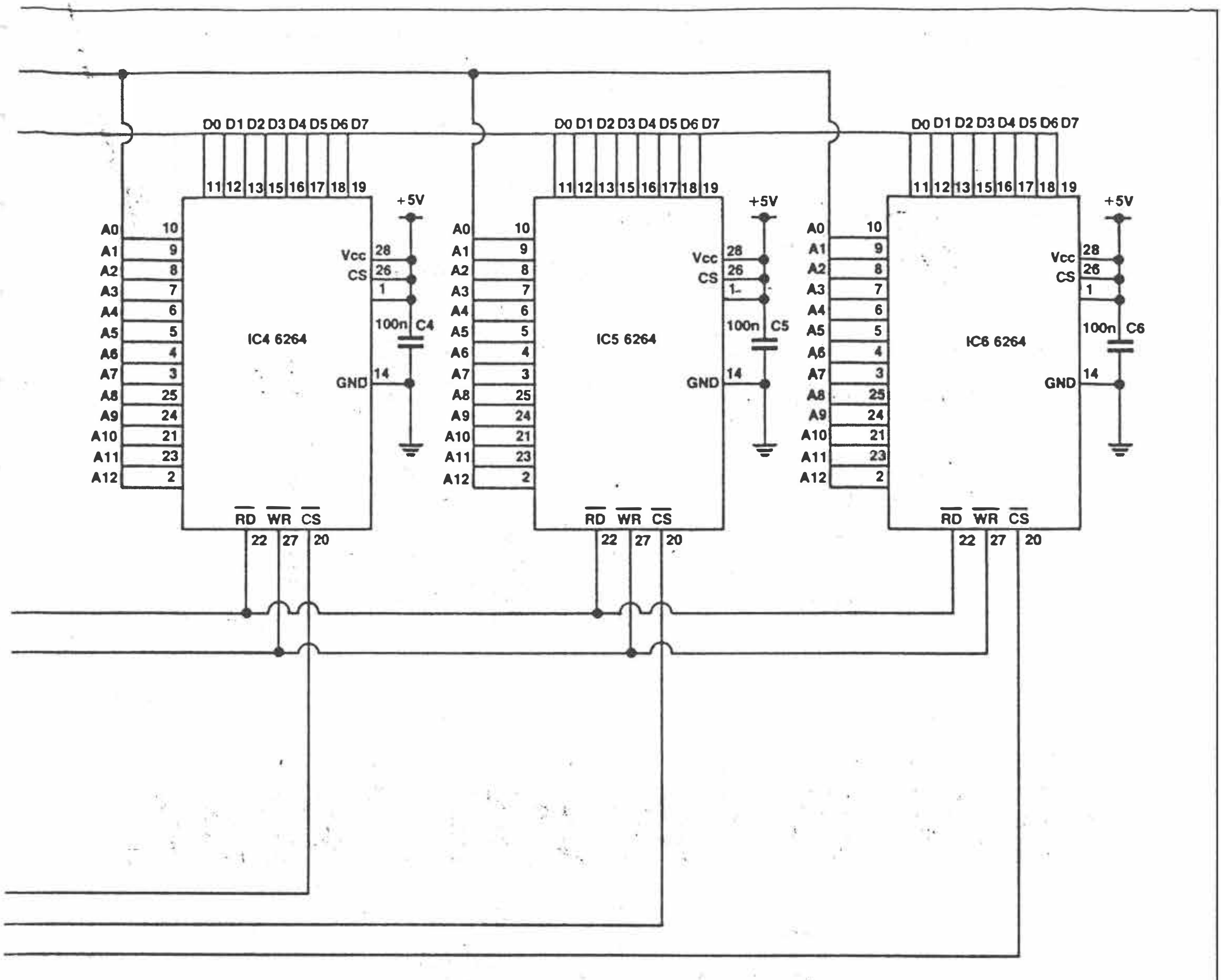
1. If the video signal wavers, then correct the clock speed by adjusting the variable capacitor by the 74LS04 on the main pcb. (See Figure 3 for the VCZ200 location. In the VZ300, the variable capacitor is located inside an rf shield to the right of the CPU. In both cases, a hole has been punched in the rf shield for access.)

2. The CPU can run at 4.433MHz by lifting pin 6 of the Z80A CPU and adding the clock circuit shown in Figure 4. The switch can be mounted on the top of the box. The speed change will affect tape operation, as this is controlled by the CPU. However, disk operations are unaffected and most programs will run somewhat faster. Do not change over the switch, however, unless the WAIT switch (see below) is depressed simultaneously, or no power is applied. The Z80 does not like its clock signal to be interfered with!

3. WAIT. Providing that no dynamic RAM is used (apologies to VZ300 and commercial RAM module users as these depend on the CPU for refresh signals), a pushbutton between pin 24 of the Z80 and ground will cause the CPU to halt while the key is depressed. Not only will this help the speed change circuit above, but the can be stopped at any time (even at a critical stage in a game!) without affecting the software (except that dynamic RAM will clear).

4. RESET. Locate the 10μF capacitor on the 74LS04 (see Figure 3; this is the same IC as is used for the clock signal). Connect a pushbutton across this capacitor. When the button is depressed, the capacitor will discharge, causing a reset signal on the CPU. This has the advantage of resetting without

4 of 6.



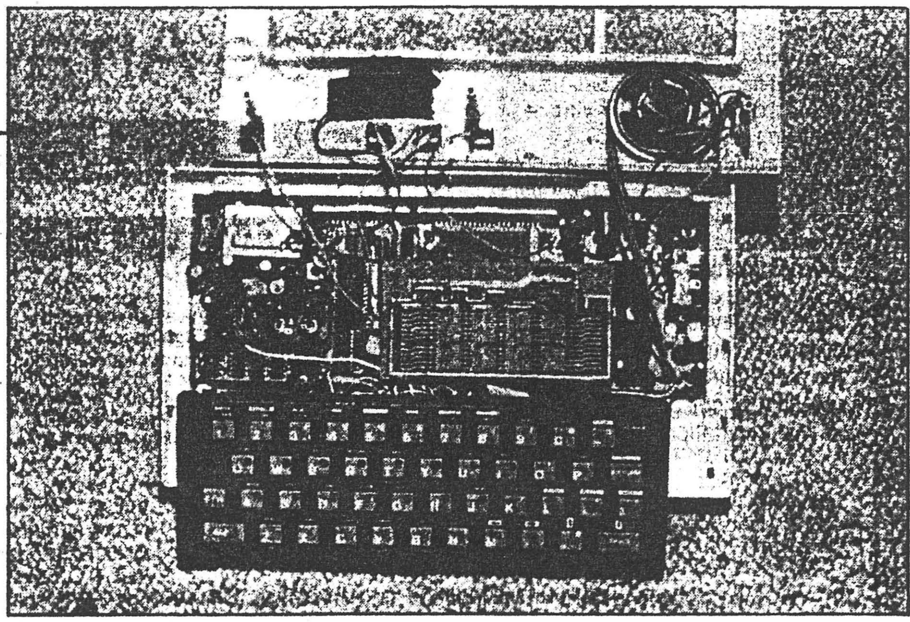
KEYBOARD SOCKET CONNECTIONS

Socket: Pin Number:	SK7/SK9/SK11	SK8/SK10/SK12
1	A15	$\overline{MREQ}$
2	A13	A12
3	A14	A11
4	+5V	A7
5	A8	A6
6	A9	A5
7	$\overline{WR}$	A4
8	$\overline{RD}$	A3
9	A10	A2
10	$\overline{78XX}$	A1
11	D7	A0
12	D6	D0
13	D5	D1
14	D4	D2
15	D3	GROUND

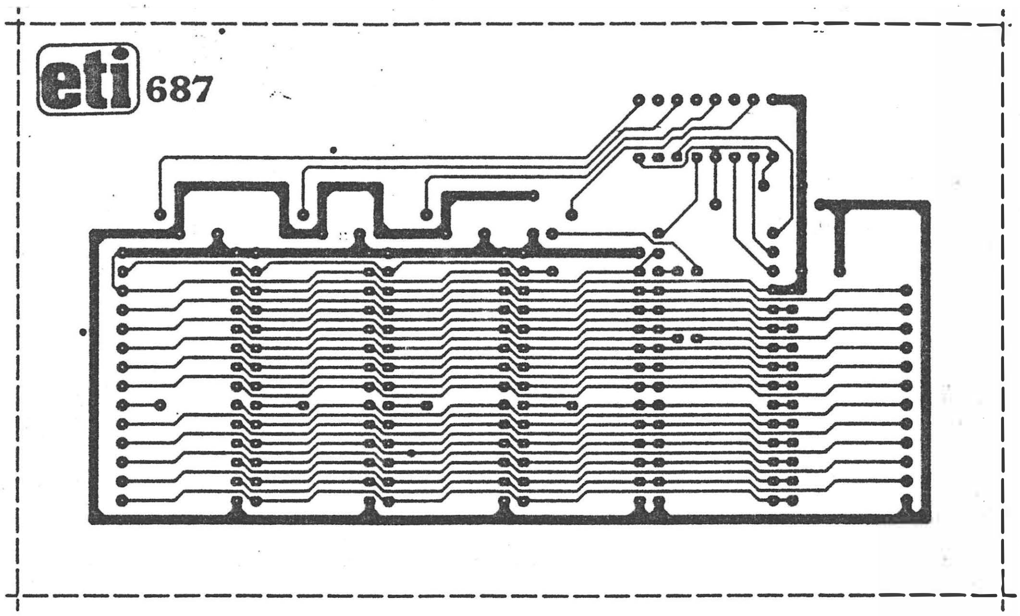
losing the memory, although all the program pointers will need to be adjusted. It is also somewhat kinder to the computer than turning off and on again.

5. Power supply. I have found that the power supply runs far too hot for my liking. Therefore, I mounted a 78H05 on a heatsink on the top right of the case, removed the present 7805 and heatsink, and wired the 78H05 in its place. Not only does the computer now run cooler, it no longer packs up when all my peripherals are connected!

The memory board mounted upside down inside the VZ200 case.



5 of 6.



6 of 6.

NOTES & ERRATA

Project 687, VZ200 modification, July '86: Pin nos 4 and 5 of IC1 were transposed. They should be as outlined below.

Computer drive for the EA EPROM Programmer

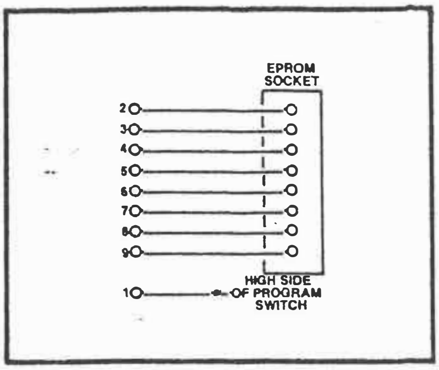
The following is a modification to the Free-standing EPROM Programmer (EA, January 1982)) to enable it to be driven from a Centronics printer port. Included are program listings for the VZ200/300 and TRS80 Models III and IV. A printer interface is required for the VZ200/300.

The hardware modifications are quite simple and mainly involve connecting the Centronics socket to the D0-D7 pins on the EPROM socket and to the high side of the program switch as shown in Fig.1. In addition, the copper tracks at pin 1 of IC5 and pins 1 and 2 of IC4 should be cut and a DPDT switch wired across the breaks.

This new switch allows the EPROM programmer to be switched to either external drive mode or to stand-alone mode.

To operate with computer drive, set the added switch to EXTERNAL, set switch S1 to WRITE, S2 to AUTO INC., and S3 to PROGRAM READY. Now load and run the program. You will have to enter the start address for data to be sent to the programmer and enter the end address.

The program takes care of most user mistakes. However, if data being sent to the EPROM is long enough to cause the address counter to reset while data is still being sent, all data sent after reset will be programmed into EPROM address 000.



Rick Buhre,
Mackay, Qld.

Rick Buhre.
41 Mogford St.,
Mackay, Queensland,
4740. Australia.
PHONE (079) 522959


```
10 CLS:PRINT"***********************************"
20 PRINT"*    EPROM PROGRAMMER DRIVER      *"
30 PRINT"*            BY R.J.BUHRE         *"
40 PRINT"*           COPYRIGHT 1986        *"
45 PRINT"***********************************"
50 FORX=1TO2000:NEXT:CLS
60 INPUT"START ADDRESS IN HEX";C$
70 IFC$=""ORLEN(C$)>4THENPRINT"INPUT ERROR!!!!":GOTO60ELSEGOSUB220
80 GOSUB270
90 INPUT"END ADDRESS IN HEX";C$
100 IFC$=""ORLEN(C$)>4THENPRINT"INPUT ERROR!!!!":GOTO90ELSEGOSUB220
110 GOSUB280
120 IFA>BPRINT"START GREATER THAN END!!!!":GOTO60ELSE140
140 FORI=ATOB:IFI>32767THENK=I-65536ELSEK=I
150 OUT14,PEEK(K):OUT1,1:PRINT@268," ";PEEK(K);"  "
160 NEXTI
170 CLS:PRINT"ADDRESS BLOCK PROGRAMMED"
180 INPUT"DO YOU WANT TO PROGRAM ANOTHER  BLOCK (Y/N)  ";A$
190 IFA$="Y"THEN60ELSE200
200 INPUT"DO YOU WANT TO PROGRAM ANOTHER EPROM (Y/N)  ";B$
210 IFB$="Y"THEN60ELSECLS:END
220 C=0:FORL=LEN(C$)TO1STEP-1
230 D$=MID$(C$,L,1)
240 IFD$>"@"THENM=ASC(D$)-55ELSEM=VAL(D$)
250 IFM>0THENN=M*(16^(LEN(C$)-L))ELSEN=0
260 C=C+N:NEXTL:RETURN
270 A=C:RETURN
280 B=C:RETURN
```

(Obtained from Author. Jan/Feb 87)
– see letter file.

MORSE INTERFACE

Arthur Forster VK2DKF
5 Hersey Street, Blaxland, NSW. 2774

This Morse interface circuit can clean up noisy Morse signals copied from a HF receiver.

It provides a "clean" processed output signal at TTL level, or a constant tone for feeding to cassette or the cassette input of a personal computer.

Many amateurs and SWLs have software programs that enable them to copy Morse from a communications receiver and display it on their personal computer. There are many hardware interface circuits for RTTY available to constructors, but very few interfaces to copy Morse. The writer has found that the simplest interfaces are not satisfactory when trying to copy Morse on a computer from the HF bands. Any noise spikes present on the signal are usually interpreted by the computer as dots and the print-out contains mostly garbage.

When training, the human ear can copy Morse code which is partly masked by noise, interference from adjacent signals and fading. The computer however, has not this level of intelligence. One other area where the human ear is superior to the computer is in the spacing of the dots and dashes. If the correct spacing is not maintained by a hand keyer the computer will not be able to copy properly, irrespective of this interface.

In principle, the function of this circuit is to provide a sharp narrow band filter, followed by an audio tone decoder. Although the filter will provide good selectivity to interfering signals, it is not sufficient for pulse-type noise which has a relatively large bandwidth. Hence the signal is further processed by applying it to a tone decoder, integrator and comparator.

CIRCUIT DESCRIPTION

This interface consists of two parts:

1. A sharp audio filter centred on approximately 800 Hz.
2. A tone decoder and processor circuit.

The audio filter is composed of an input buffer stage IC1, followed by a four stage active filter, IC2, IC3. This filter gives very sharp rejection to any signals either side of its centre frequency. It is very useful when decoding a signal very close to unwanted signals.

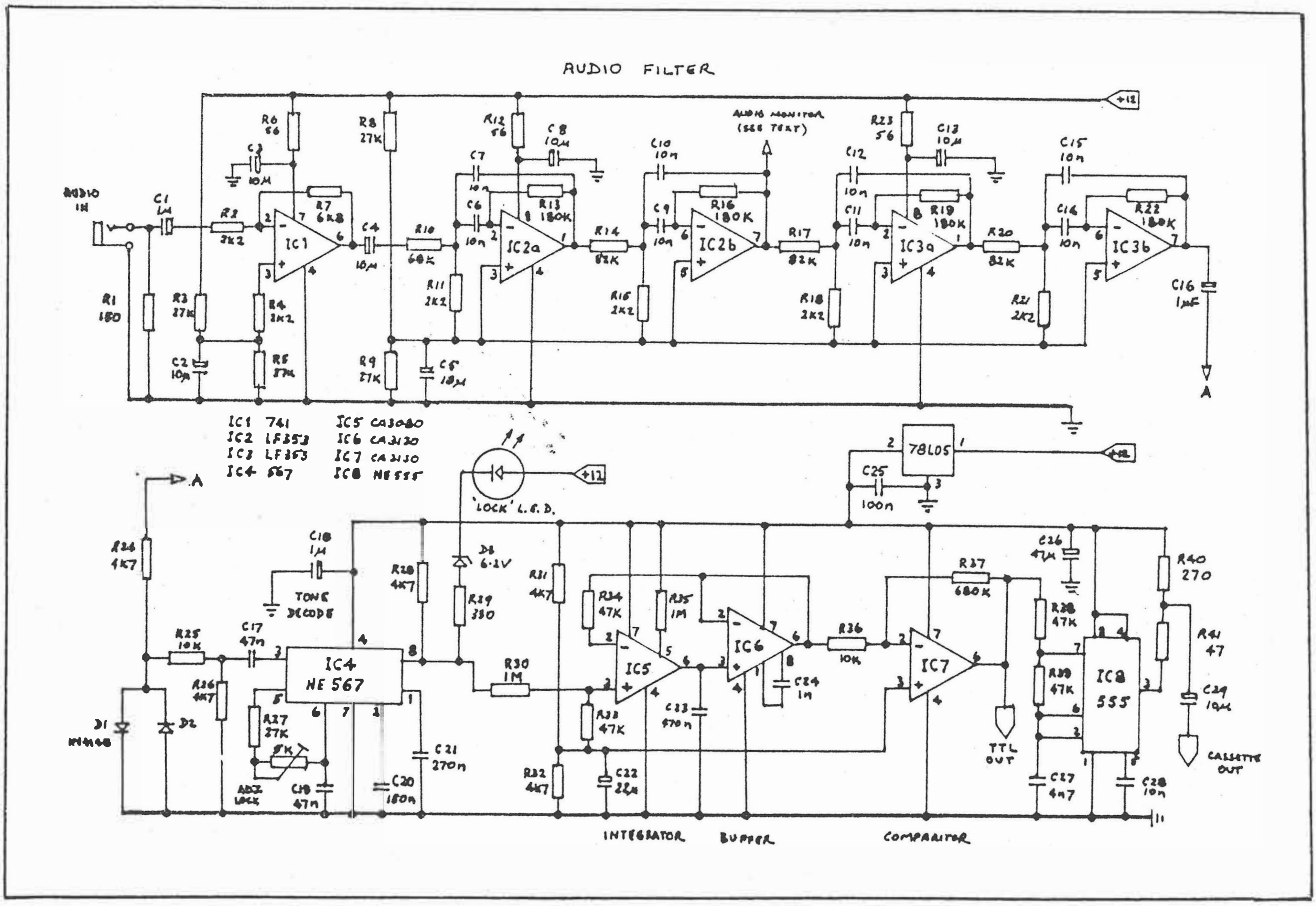
The output of the filter is then fed via a resistive attenuator network to the input of the Tone Decoder, IC4, on the second board. The back-to-back diodes ensure that the input signal level is limited to 600 mV peak-to-peak.

The frequency of the Tone Decoder IC4, is set precisely to the filter centre frequency by R27, C19 and preset potentiometer. The output of IC4 at pin 8, goes to logic 0 as soon as a 800 Hz signal is applied to its input, causing the lock LED to light. However, the Tone Decoder also responds to short interfering noise spikes that pass through the earlier filter. These pulses are eliminated by the following circuit consisting of IC5, IC6, IC7.

IC5 is configured as an integrator whose time constant is determined by the control current flowing via R35 into pin 7 and by capacitor C23. This has the effect of eliminating short pulses. IC6 is a voltage follower to prevent loading on integrating capacitor C23. IC7 is configured as a comparator with a threshold voltage of 2.5 volts.

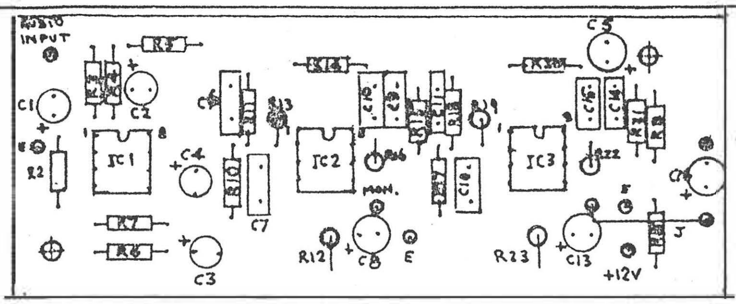
The output from pin 6 of IC7 will be at TTL level, going between 0 volts and +5 volts, depending on whether a tone (dot, dash) is present or not. This output can be used to interface with the input port of a computer that requires a TTL input.

The writer designed this interface for use with a software program for the VZ200/300 that requires an audio tone input to the cassette input of the computer. Therefore, IC8, an NE555 timer, is configured as a square-wave

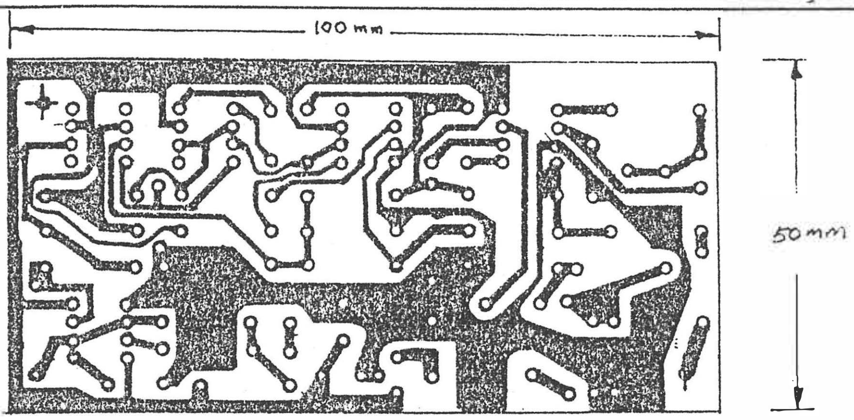


1 of 2

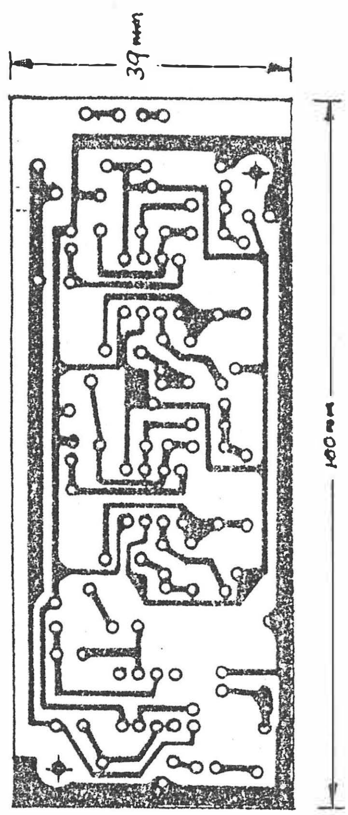
p. 16-17.



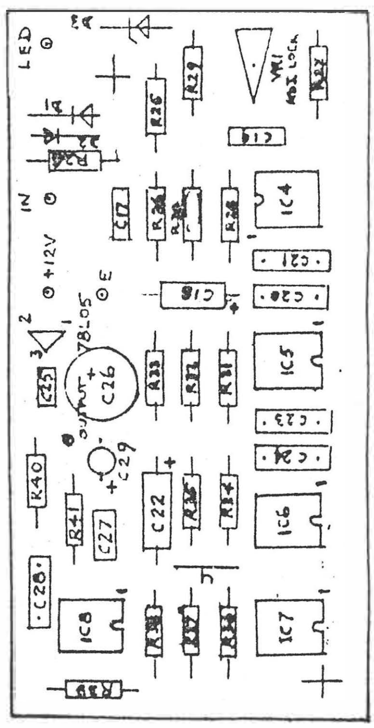
Filter Board.



Board Layout.



Copper Track Side.



Processor Board.

tone oscillator. The preceding stage switches the tone on and off by switching the voltage on pin 7 of the IC. The output level at pin 3 is adjusted by R40, 41 to give the correct level into the cassette input of the computer.

If an audio monitor point is required, it could be taken from the output of IC8 but a better point would be from pin 7 of IC2 in the CW filter. The monitor signal could be buffered by a simple IC audio amplifier as per Figure 3 and brought out to a socket to drive a speaker or headphones. The circuit is supplied from an external 12 volt source that could be a DC plug-pack. The +5 volts rail is derived from the +12 volts rail very simply by using a 78L05 low power regulator transistor.

CONSTRUCTION

The circuit was laid out on two separate printed circuit boards to ensure as much flexibility as possible. The nature of the case housing the circuitry is left to the discretion of the constructor. The writer was able to mount the boards in the same case that contains a RTTY interface and thus obtain a single compact modem that can be used for CW as well as RTTY. Audio input and computer output connection are by way of miniature 3.5 mm jack sockets.

It is important to use close tolerance resistors and capacitors in the feedback circuits around IC2, IC3 of the CW filter. Preferably the capacitors could be checked using a capacitance bridge. Signal leads between the boards and the output sockets should be wired in shielded cable.

As some of the ICs are FET devices, the usual precautions against static damage should be observed. They were mounted directly on the printed board without sockets in the prototype, with the usual precaution of soldering the earth and supply pins first, using a properly earthed soldering iron.

ALIGNMENT AND USE

There is only one adjustment to be made after the unit has been constructed and the supply voltages checked to see that it is functioning correctly.

First check that the voltage on the input bias pins of the ICs is approximately half the rail voltage. Connect the audio input of the modem to the headphone output socket of a HF receiver and tune in a CW signal accurately so that the "Lock" LED lights in sympathy with the incoming CW signal. Reduce the receiver's audio volume control to a level where the LED just lights and adjust the preset "Lock" potentiometer for the minimum level of audio from the receiver that still keeps the circuit in lock. This will be the point where the tone decoder's frequency is adjusted to the centre point of the CW filter.

Check that a tone of approximately 1 kHz is being switched on and off at the output of IC8.

In use, it will be found that the circuit is quite sensitive and the audio input should be kept reasonably low so long as the decoder still stays in lock, indicated by the lock LED lighting at full intensity.

In operation, the circuit makes a surprising difference when listening to noisy signals. It could be used without a computer for monitoring off-air signals under difficult reception conditions.

MORSE SOFTWARE PROGRAM

The writer is using a machine code Morse program written by Ross ZL1BNV, for the VZ200/300 computer.

This program has such features as sending and receiving with a speed rage of 1 to 99 WPM and split screen display. Input and output is via the computer's cassette I/O port.

PARTS LIST

RESISTORS: ½ watt 5 percent

R1	150 ohm
R2, 4, 11, 15, 18, 21	2k2 ohm
R3, 5, 8, 9, 27	27k ohm
R6, 12, 23	56 ohm
R7	6k8 ohm
R10	68k ohm
R13, 16, 19, 22	(2 percent) 180k ohm
R14, 17, 20	(2 percent) 82k ohm
R24, 26, 28, 31, 32, 33, 34, 38, 39	4k7 ohm
R29	330 ohm
R30, 35	1M ohm
R25, 36	10k ohm
R37	680k ohm
R40	270 ohm
R41	47 ohm
R42	(preset pot) 5k ohm

2 of 2.

16K Memory for VZ-300 Computer

This 16K expansion can be built for considerably less than commercial versions. It comprises two 8K x 8 6264 CMOS static RAMs, a 74HC138 1-of-8 decoder and a 4008 4-bit adder.

IC3 and IC4 provide decoding of the A11 to A14 memory addresses to select IC1 and IC2 via the CS1-bar chip select inputs. The Y0 and Y1 outputs of IC3 ensure that when IC1 is selected IC2 is deselected and conversely, when IC2 is selected IC1 is deselected. A15 is used to select both IC1 and IC2 via the CS2 chip selects.

The MREQuest-bar line is used to enable IC3 via the G2A-bar and G2B-bar inputs.

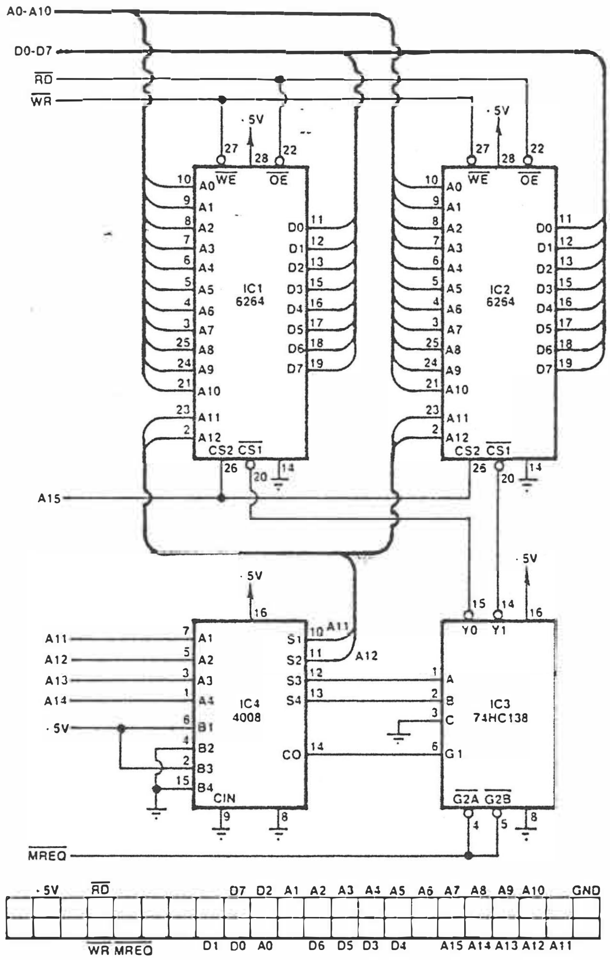
Read and Write (RD-bar and WR-bar) lines select the Write Enable-bar (WE-bar) and Output Enable-bar (OE-bar) of both IC1 and IC2.

Data lines D0 to D7 connect to the D0 to D7 lines of both IC1 and IC2. For the memory, A0 to A10 connect directly to the A0 to A10 lines of IC1 and IC2, while A11 and A12 connect via IC4.

Construction can be wire wrap or on Veroboard. A 44-way 2.54mm (0.1 inch) edge connector connects the memory expansion to the VZ-300 computer. The connections for this bus are shown.

M Kosovich,
Midland, WA.

$20



1 of 3

Notes & Errata

VZ-300 MEMORY EXPANSION (May 1987, CDI). Pins 2 and 4 of IC4 should be tied low and pins 6 and 15 tied high; not 4 and 15 low, and 2 and 6 high, as indicated. Connecting the circuit as shown may cause damage to either the static RAMS or the VZ-300.

VZ-300 expansion problem

From your "Circuit & Design Ideas" in the May issue I decided to make the "16K memory add-on for the VZ-300 computer". I thought it worth taking a chance on, and at the worst I might not be able to make it work. For it to kill my computer was more than I bargained for.

Your Notes & Errata in the August issue say this might happen if the circuit is constructed in the way shown. I have changed the internal RAM chips (4116) but the fault of garbage displayed did not change. I realise that it is not your usual policy but I would be very grateful if you could comment, from advice you may have received, as to which chip or chips in the circuit are likely to have been damaged by the addition of this expansion. I hope you can help. (W.E.P., Christchurch NZ)

• ***We haven't had any further advice, but from your description that the unit now displays "garbage", it sounds as if either the 6847 video display controller chip (U15) or the 6116 video RAM (U7) may have been damaged somehow. Or perhaps the 74LS245 bus buffer U14, if there was a bus conflict. A remote possibility is that the Z80A CPU itself has been damaged. Sorry, but it's hard to offer more help than these suggestions.***

VZ 300 expansion

I would like to respond to your reply to W.E.P. ("VZ-300 Expansion Problem", Information Centre, January 1988). I think you may be on the wrong track in your advice.

The "garbage on the display" is a familiar symptom to anyone who has tried to build "add-ons" to system-80s, TRS-80 Model Is etc. I believe the VZ series has similar ROMS.

The problem is that the screen is initially cleared by the startup routine software: there is no hardware clear-screen, and until the startup routine has run, the random contents of video RAM are displayed. Hence the "garbage on the screen".

I have not seen the original circuit, so I don't know exactly what has happened. Things to check are:

(1) Are the ROMS still properly seated in their sockets?

(2) Is there a possibility that there is an address conflict between the new RAM and either the video RAM or the ROM. Perhaps try starting the computer with all RAM removed?

(3) An easy thing to do is to short an address or data line to ground or 5V, or to one of the other bus lines.

In all these cases (and in all cases I have seen) there is usually no "damage" done, no blown chips or anything. You just have to find out why the CPU is not communicating correctly with the ROM & video RAM, remove the faulty connection, and everything works again. (R.L., Downer, ACT)

• ***Thanks for the helpful advice, R.L.***

Circuit idea.

Some months ago I built a 16k memory expansion for my son's VZ-300, so far I have found it impossible to get to run propertly. The fault seems to be incorrrect memory addressing.

The circuit used came from your May 1987 magazine, in the Circuit and Design Ideas section.

Could you please tell me if any alternations or corrections were made to the circuit you published. My son is hoping to try and run Stan Blaster, which needs the extra memory, and at present is not pleased with a Dad who can't build things that work. (J.B., Nowra, NSW).

• ***Sorry J.B., but items published in the Circuit and Design Ideas section are presented "as is", directly as sent in by readers. As we note each month, we're not in a position to provide any further help with them.***

2 of 3.

A bad design?

The next correspondent raises quite a few points in a letter that has a fairly severe tone to it. The letter is in response to a previous letter concerning problems with a VZ-300 RAM expansion circuit, presented as long ago as May 1987 in our 'Circuit and Design Ideas' section. Although the circuit in question is now somewhat dated, the points raised are interesting. Here's the letter, in reduced form.

The VZ-300 RAM expansion circuit presented in the Circuit and Design Ideas (EA May 1987) section has two glaring faults. The first is that the Z80 CPU, as used in the VZ-300 has TTL level output voltages, that is, less than 0.8V (low) and greater than 2.4V (high), whereas the CMOS logic gates used in the circuit have CMOS level inputs less than 1V (low) and greater than 4V (high). Because of the incompatibility of the logic families used, it is probable the circuit will not operate correctly.

Secondly, the propagation delay of 300ns for the 4008 adder would be likely to create problems, due to the access time of the VZ-300.

Might I suggest that when checking computer circuits for feasibility, you check particularly the following points.

1. *Correct pinouts of ICs.*
2. *Correct Boolean logic.*
3. *Logic family compatibility.*
4. *Propagation delays.*

Clearly, the third and fourth points have been overlooked in the circuit, and I doubt if the designer ever actually tested his design or perhaps he got lucky with a very fast 4008 in his prototype.

Might I also suggest that you request a declaration from contributors stating that they have tried the circuit presented to save problems such as these. (M.S., Clarence Park A)

OK, the circuit referred to by M.S. is now over two years old, and delving back to it is not really going to prove anything. The reason I have published the letter is to be able to air the technical aspects of interfacing logic families, and to answer the suggestions by the correspondent on how we should check circuits presented for our Circuit and Design Ideas (CDI) section.

Examining various data books on the subject, I have to agree with M.S. concerning the likely incompatibility problems with interfacing a TTL-compatible IC to a CMOS type. The problems will arise when the TTL device goes high, and it is usual to include a pull-up resistor from the output to the 5V rail to get as high an output level from the TTL device as possible.

However, my own experience has demonstrated that most TTL compatible ICs (such as the Z80) will produce an unloaded output level of around 3.5V when the output is high. Most CMOS inputs will also recognise an input voltage of 2.5V or more as being a logic 1. So while the data books state certain limits, in practice one can often get away with interfacing TTL directly to CMOS. The simple answer is to add the pull-up resistors, which can be any value from 1k to 10k, although 2.2k is a typical value.

Propagation delays are another variable, and the times specified by manufacturers are always worst case. It often happens that CMOS ICs from one manufacturer will have different specifications to those from another, and generalising is often very misleading. For example, the Fairchild manual gives a typical propagation delay for the 4008 (at 5V) as 150ns, and 300ns as the maximum.

What I am trying to say is that I believe the circuit referred to by M.S. has every chance of working, although it does break 'good design' rules. So if I had applied the criteria suggested by M.S., this circuit would have passed my inspection, on the basis that I would not be prepared to reject it as technically inoperable because it breaks a few rules.

Then again, how on earth would we have the time to analyse all circuits presented by contributors for our CDI section, using the criteria suggested by M.S.? These circuits are presented with the disclaimer that we have not tested them – a sort of 'buyer beware' clause. Our main concern is whether the circuit is likely to be of interest to other readers. Sorry M.S., we cannot abide by your suggestions as many excellent circuit ideas would never be printed.

Finally, I doubt if a signed declaration by contributors confirming that they have tested their circuit would solve anything. All the signing in the world simply means the prototype worked, which may be the result of good luck, or it may mean considerable research to ensure repeatability has been undertaken – who knows? Also, I question whether contributors would bother to dream up a circuit that they never actually built and submit it for publication anyway. We take the attitude that most contributors are honest, and our disclaimer takes care of the rest.

3 of 3.

VZ200/300 software

Dear Sir,

I am writing to you to see if the software is available for the Listening Post and Project 3503 to suit the VZ200/300. You have indicated previously that suitable software might be published sometime.

I find it very hard to get software for my computer in the area of amateur radio. I am looking for software useful for DX, antenna design and propogation predictions as well as satellite data.

R. Thompson
Gorokan, NSW

In response to your question about Listening Post software, we are still trying to find someone who can re-write the Microbee program to make it suitable. While both the Microbee and the VZ200/300 employ a Z80 microprocessor, their internal "architecture" is different. The Microbee program also calls routines resident in its ROM and the VZ doesn't have these.

A further complication arises with the VZ in that it does not have any accessible ports other than the Z80 bus expansion so it may be necessary to provide some decoding hardware as well as adapting the software. We will keep you informed of progress and would be very happy to hear from anyone with VZ200/300 experience who might like to attempt the job.

As far as amateur radio software is concerned, there are a number of good books available which supply listings of programs for most aspects of amateur radio. Most of these programs are in BASIC and should run with very few changes on the VZ200/300. We have had a number of enquiries about satellite software and we are currently working on some suitable material which we hope to publish in the near future.

Andy Keir.

MEMORY EXPANSION FOR THE VZ200/VZ300 COMPUTERS

Lloyd Butler VK5BR
18 Ottawa Avenue, Panorama, SA. 5041

The unit described extends the memory of the VZ200 by 20 k bytes and the VZ300 by 18 k bytes.

IF YOU OWN a VZ200 or VZ300 computer, you could be interested in extending the memory to run larger programs. To do this, you may choose to visit the nearest Dick Smith store and purchase a memory expansion module. Alternatively, you may take the second option and build one yourself.

The writer decided on the second option and designed the unit described in this article. Making use of the 8 k static RAM packages, now readily available, assembly of the unit was a straightforward task.

DESCRIPTION

Two 8 k static RAM packages, Type 6264, provide 16 k bytes of additional memory. To simplify decoding of memory chip selection, the start locations of the 8 k RAM packages are connected at precise 8 k (or 2000 H) address multiples within the address range. Because the in-built memories of the VZ200/VZ300 do not end just prior to such locations, one additional 2 k RAM Type 6116 is used to fill in the gap at the end of the VZ300 internal memory and two at the end of the VZ200 internal memory. For the VZ300, the memory is therefore extended by 18 k bytes. (This, with the in-built system ROM and in-built RAM, utilises all of the 64 k address range of the VZ300 computer). For the VZ200, the memory is extended by an additional 20 k bytes.

The wiring diagram for the expansion unit is shown in Figure 1. The 8 k RAM packages (28 pin DIL) are shown as N3 and N4 and the 2 k RAM packages (24 pin DIL) as N5 and N6. Chip select decoding is carried out by two 74LS138 decoder packages (16 pin DIL) shown as N1 and N2. A five volt regulator, N7, is included in the unit to supply power to the IC packages. This was thought desirable as total loading on the internal five volt supply might have been marginal with the extra load of the expansion unit.

A three pole, two position, switch (S1) is provided to select decoding for either VZ200 or VZ300. (The switch used was a four pole unit with one redundant section). If only the VZ300 facility had been required without the VZ200, the 2 k RAM (N6), resistor R1 and the switch, could have been omitted. In this case, switch connections S1A and S1B for the VZ300 would be bridged.

The hexadecimal start addresses for the RAM packages are shown in the following table with the decimal addresses, as identified by the BASIC interpreter, shown in brackets.

PACKAGE	VZ200	VZ300
N5 (2 k)	9000 H (-28672)	8800 H (-18432)
N6 (2 k)	9800 H (-26624)	not used
N3 (8 k)	C000 H (-16384)	C000 H (-16384
N4 (8 k)	A000 H (-24574)	E000 H (-8192)

The complete memory map, with expansion unit included, is illustrated in Figure 2.

A further option for the VZ200 (but not used by the writer) could be to parallel up the buses for a third 8 k 6264 RAM to be started at E000H. This would then extend the VZ200 also to the full 64 k capacity. All that would be required for additional chip selection would be to connect the RAM chip select (pin 20) via a switch circuit (similar to S1C) to pin 7 on decoder N1.

ASSEMBLY

The assembled module card is shown in Figure 3. A general purpose circuit board was used to mount the IC sockets and other components. There are various types of board, with printed circuit pads for solder connections, which can be used to do the job. Another method would be to make use of wire-wrap with wire-wrap type IC sockets.

The card was cut to the dimensions 145 by 92 millimetres. It could have been made smaller but allowance was made for components to be added had they been needed. (This is a practice which often pays off on a first attempt at a design).

A 69.5 millimetre length of 0.1 inch (2.54 millimetres) pin spacing edge connector was fitted to the card. The edge connector was carefully cut so that the 22 pairs of pins used are centred to mate with the printed circuit edge pins on the VZ memory expansion connector and so that the edge connector is correctly guided by the recess in the VZ case. The fitting of the edge connector to the circuit board is offset so that it clears the I/O expansion entry. The method of assembly is similar to that previously used by the writer in the RTTY/Morse module described in *Amateur Radio,* September 1985 and January 1986.

A light aluminium box, 96 by 156 by 24 millimetres, was constructed and fitted around the card for protection. The connector protruded through the end of the box so that it could project into the VZ connector recess.

CHECKOUT

Having made sure all the wiring was correctly routed by carrying out a continuity check, the next step was to devise a functional check routine and a program in BASIC was prepared to check out the additional RAM. This is listed in the Appendix.

For each memory address, the program write zeros into all bits and then reads the address to check for concurrence. The process is repeated for ones in each bit and then again for zeros. The memory is accessed sequentially over the whole extended range and, if an address does not read as written, the sequence is stopped and the address identified. The option is then given whether to proceed or escape from the routine. If all memory addresses check out, the memory is flagged as "OK".

At the start of the program there are POKE statements which shift the location of the top of the memory pointer and the stack pointer to within the internal memory. This is necessary as, at power up, the inbuilt VZ monitor automatically searches for the top of memory and references to these pointers to the top part of the expansion memory about to be accessed. If not relocated, the program will "crash" when it gets near the top. Actually there are two separate routines. The first one, which resets the pointers, is started by a RUN command. At its end, this routine requests a RUN 20 command which is used to start the next routine containing the memory scanning process. One might think that it could all be done in the one routine but the writer could not get it to work that way!

The inbuilt BASIC interpreter is comparatively slow and to run this program through the full 20 k bytes of additional memory takes about three-quarters of an hour. (It is a good plan to go away and make a cup of coffee while it is all going on!). Preparation of an object deck would have speeded up the process but this was not considered warranted for the few times the program was to be used.

CONCLUSION

Use of the 8 k static RAMs provides a simpler circuit design than that of the stock dynamic RAM expansion unit published in the VZ200 Technical Reference Manual. The static RAMs are expensive but, providing one does not mind spending a little time on construction, the unit described can be considered to be reasonably cost effective as well as providing a little more memory than the stock unit.

APPENDIX

Expansion RAM Test Program

```
10  REM EXTENSION MEMORY RAM
    CHECK
14  POKE 30880,255:POKE 30881,141
```

1 of 5

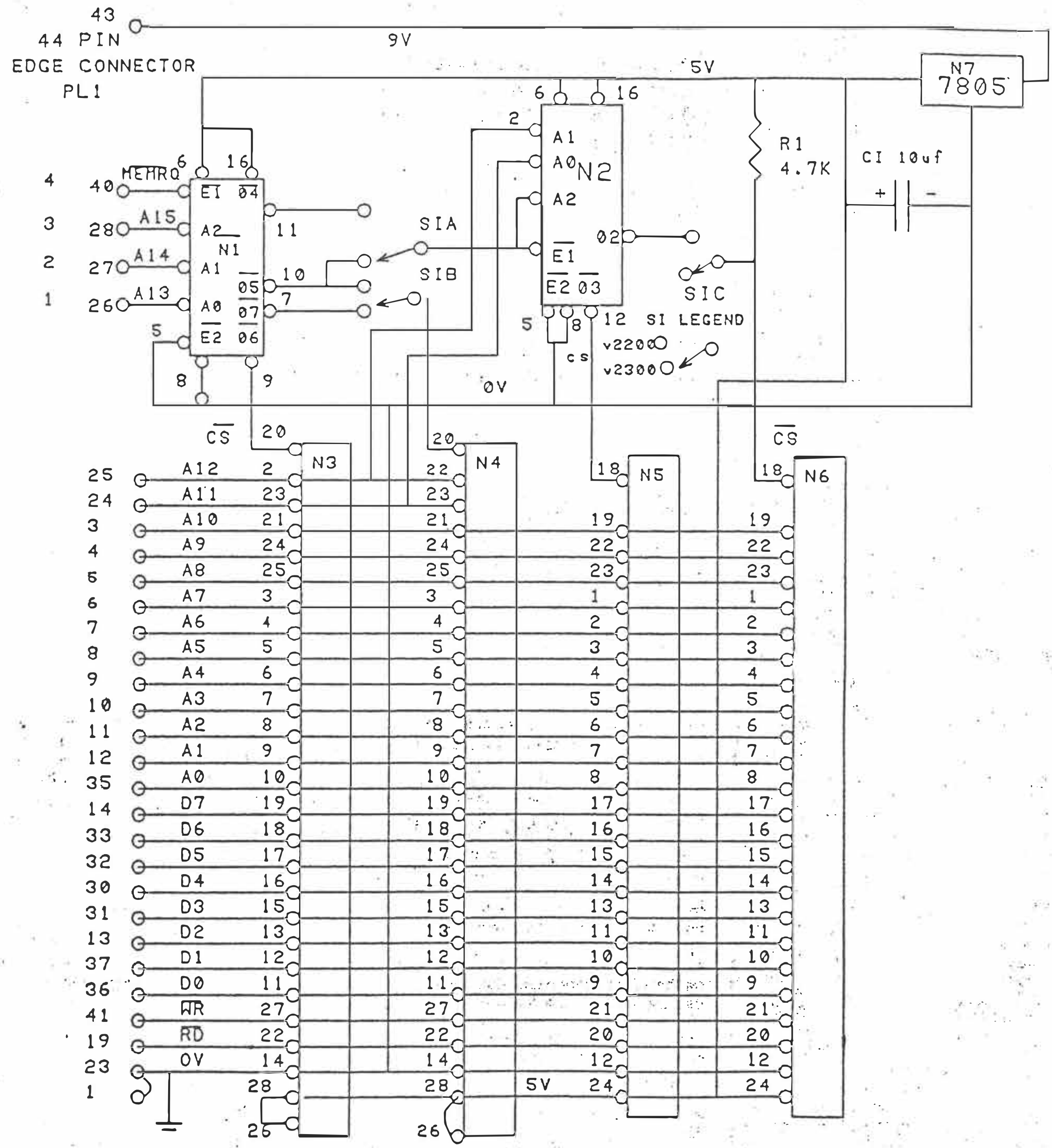
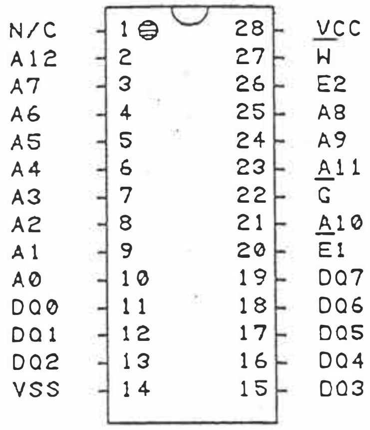
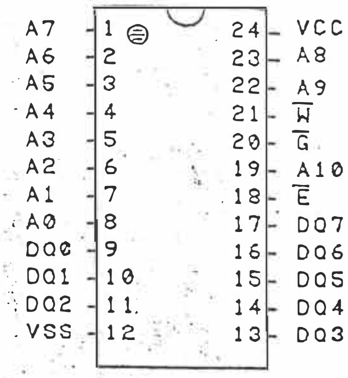


Figure 1: VZ200/VZ300 Expansion Module — Wiring Diagram.





PIN NAMES	
A0-A12	ADDRESS
W	WRITE ENABLE
E1.E2	CHIP ENABLE
DQ0-DQ7	DATA INPUT/OUTPUT
VCC	+ 5V POWER SUPPLY
VSS	GROUND
G	OUTPUT ENABLE

PIN NAMES	
A0-A10	ADDRESS INPUT
DQ0-DQ7	DATA INPUT/OUTPUT
W	WRITE ENABLE
G	OUTPUT ENABLE
E	CHIP ENABLE
VCC	POWER +5V
VSS	GROUND

6264

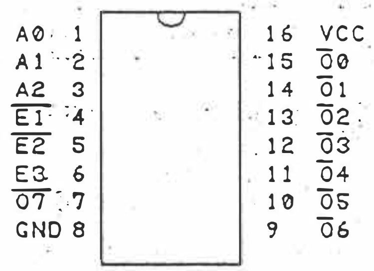
6116

N1-N2 74LS138
DECODER

N3-N4 6264
8K*8RAM

N5-N6 6116
2K*8RAM

C2-C7 0.1uF
CONNECTED ACROSS
5V RAILS AT EACH
I/C N1-N6



PIN NAMES	DESCRIPTION
A0-A2	ADDRESS INPUTS
E1.E2	ENABLE INPUTS (ACTIVE LOW)
E3	ENABLE INPUTS (ACTIVE HIGH)
O0-O7	OUTPUTS (ACTIVE HIGH)

74LS138

FIGURE 1

V2200/VZ300 EXPANSION
MODULE WIRING DIAGRAM

3 of 5

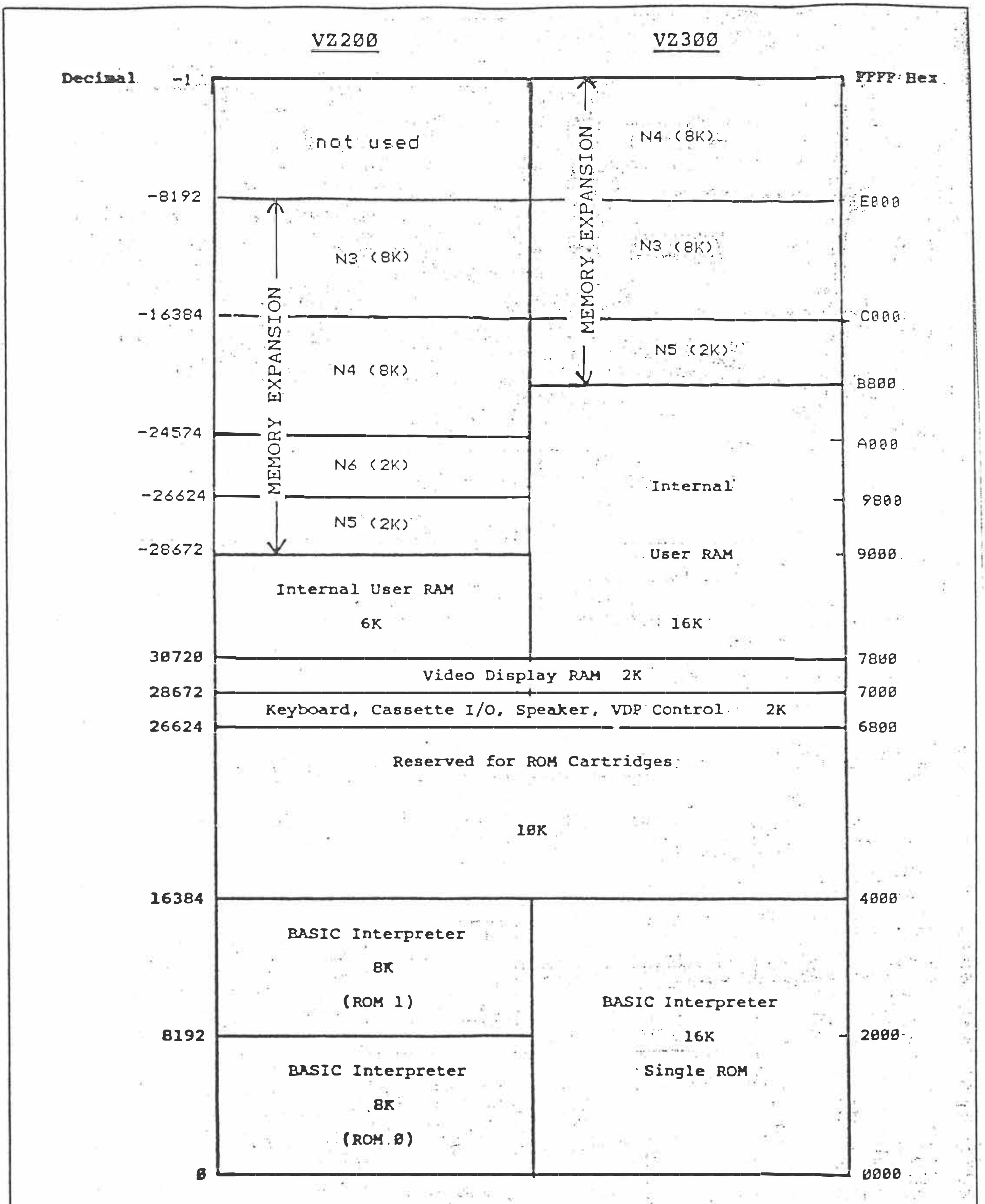


Figure 2: Memory Map showing Expansion RAM.

4 of 5.

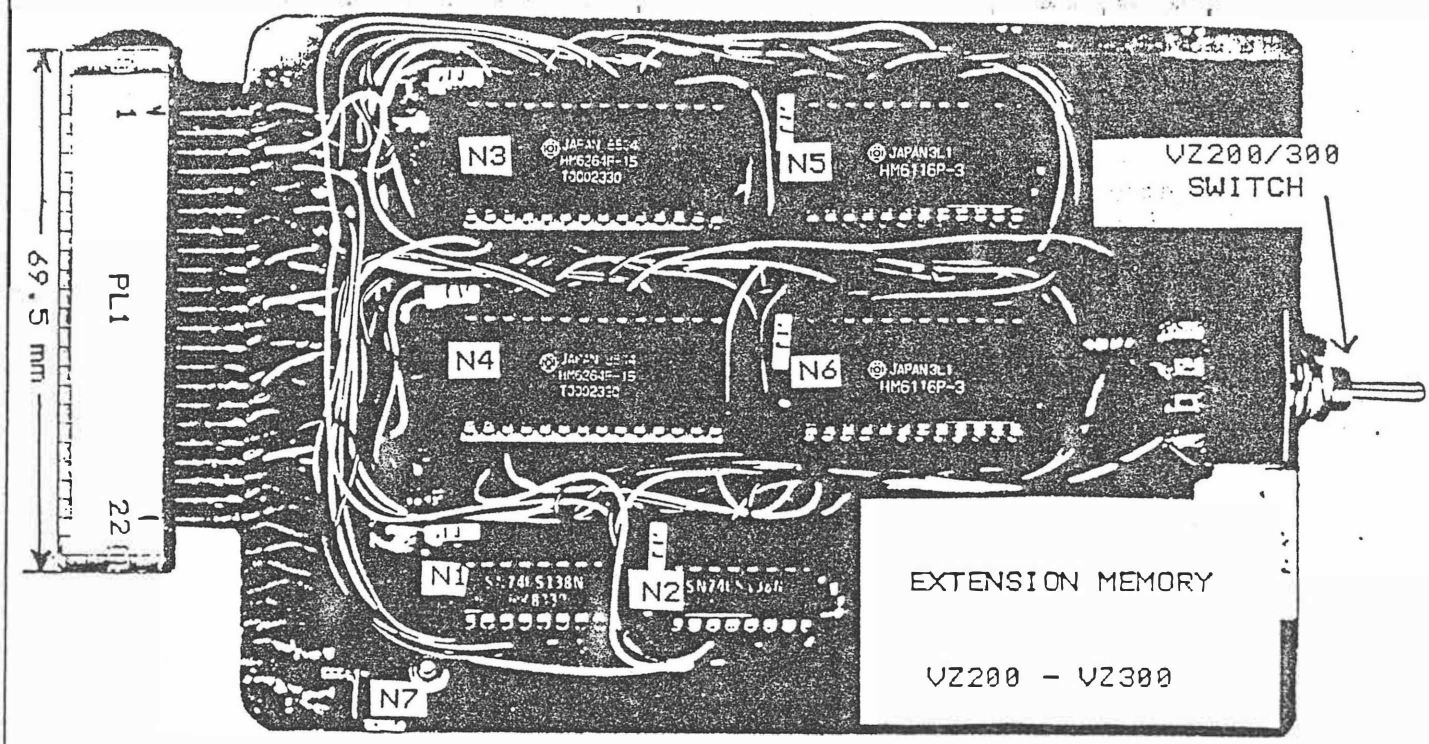


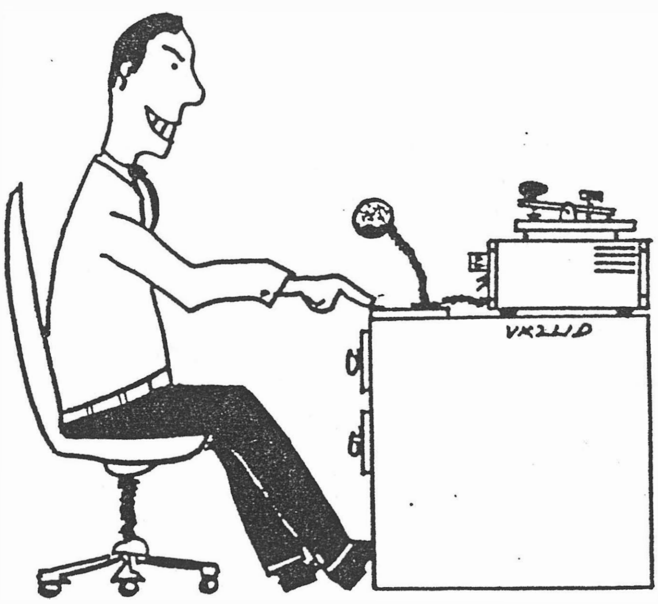
Figure 3: Card Layout.

```
15   POKE 30897,255:POKE 30898,143
16   PRINT "ENTER RUN 20"
17   END
20   PRINT "EXTENSION MEMORY TEST"
30   PRINT "ENTER 200 FOR VZ200 OR 300
     FOR VZ300
40   INPUT A
50   IF A = 200 THEN S = -28672 ELSE S =
     -18432
60   IF A = 200 THEN F = -8193 ELSE F =
     -1
70   L = 0
80   FOR X = S TO F
90   I = 0
100  FOR Y = 1 TO 3
110  IF Y = 2 THEN K = 255 ELSE K = 0
120  POKE X,K
130  B = PEEK(X)
140  IF B < > K THEN I = 1
150  NEXT Y
160  PRINT X
170  IF I = 0 THEN GOTO 230
180  L = 1
190  PRINT "RAM FAULT AT "; X
200  PRINT "ENTER C TO CONTINUE
     CHECKS OR E TO END"
210  INPUT Z$
220  IF Z$ = "E" THEN GOTO 250
230  NEXT X
240  IF L = 0 THEN PRINT "EXTENSION
     RAM OK"
250  END
```

ar

"CQ DX — New Countries only, please!"

—VK2COP



An "ultra-graphics" adaptor for the VZ200/300 computers

Matthew Sorell

Are you sick of the graphics and text restrictions on your VZ200/300? Then this project is for you. Offering 256 new characters, including upper and lower case, Greek, DATA70, mathematical and other symbols, as well as graphics up to six times the normal resolution, the Ultra-Graphics extension board is a must for the serious VZ200/300 owner.

INSIDE THE VZ computer lies a very versatile video IC. Unfortunately, the designers were working on a low budget machine and so the graphics capabilities are quite limited. However, by extending the amount of video RAM used, adding a character generator EPROM and a few other ICs, the graphics capabilities of both the VZ200 and VZ300 can be fully realised.

The first problem, then, is to fit 6K of RAM into a 2K memory position. To do this, a latch was used to provide an extra two address bits to bank switch an 8K RAM into the normal 2K of video RAM space, in position 7000-77FFH (28672-30719). As an 8K RAM is used, but the highest resolution available only uses 6K, an extra 2K of general data storage RAM is available. This can be used, for example, to store character definitions for use in high resolution graphics.

The latch used was installed into I/O address 20-2FH (32-47), which is the same position as the joystick controller. However, as the joystick is a Read-Only device, a Write-Only Latch will not interfere with it. The latch also controls the new graphics and text modes.

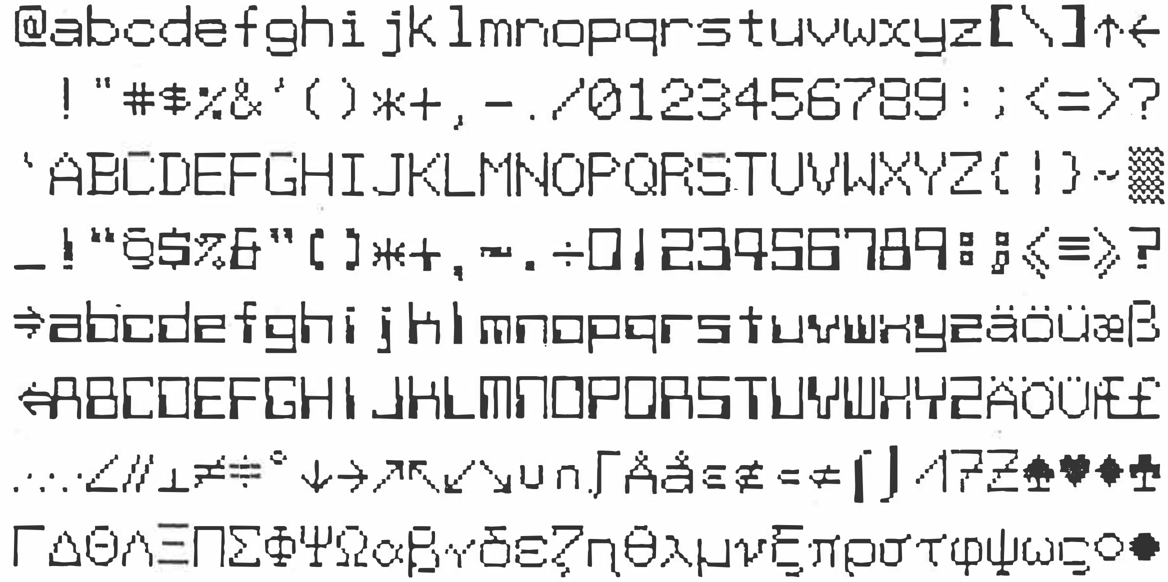
A word of warning: This project is an extensive internal modification to the VZ200 or VZ300 computers. If you are not confident about modifying the computer, then I recommend you do *NOT* attempt this project without experienced help. I also strongly recommend you obtain a copy of the "VZ300 Technical Manual", which will assist you if problems arise. Building this project also voids the manufacturer's warranty, so it's best tackled after your machine's warranty has expired.

New characters, extended graphics

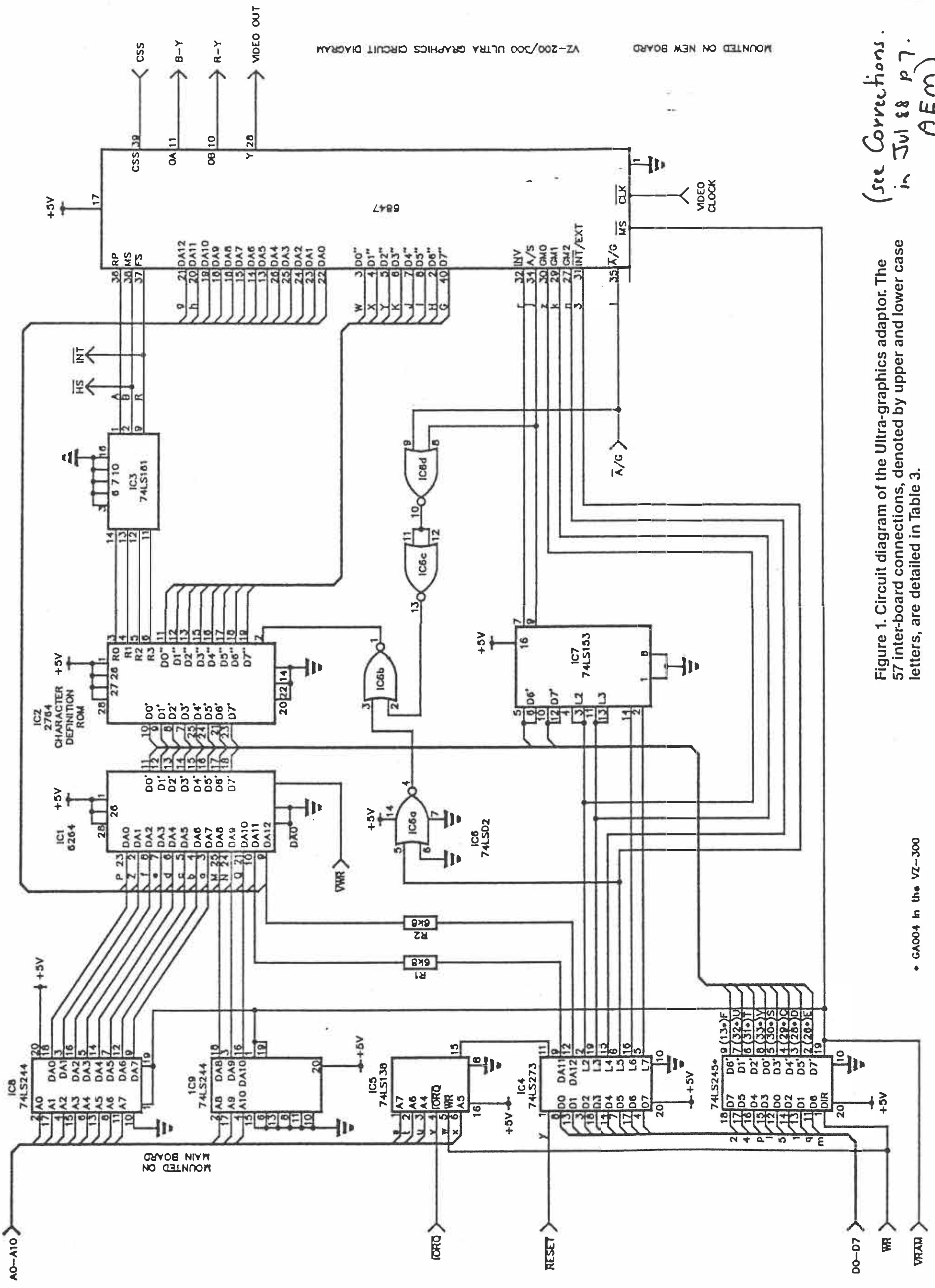
The new character set is shown in Chart 1 here. It was originally designed to be compatible with the VZ word processor (tape version). Thus there is the 96 standard ASCII characters, which are slightly out of order to be more compatible with the standard VZ text. There is also a DATA70 ("computer" type) character set, a Greek character set, some international characters, and mathematical symbols which can be accessed by poking their code into video memory, or printing the correct semigraphics character in the right colour. A dedicated screen controller routine could also be used.

The new graphics modes serve many useful purposes. The highest resolution graphics mode (256 x 192 pixels), is equivalent to the resolution in text mode, and so can be used either for text, using a suitable driver routine, or for graphics,

CHART 1. The new character set. Note the addition of special symbols, Greek and maths symbols and Data 70 characters.



1 of 8.

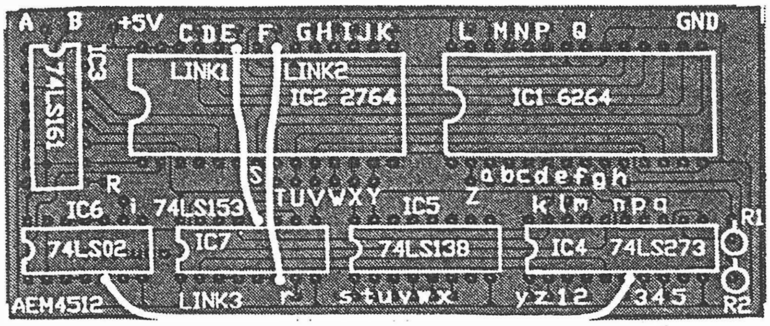


(see Corrections in Jul 88 p7. AEM)

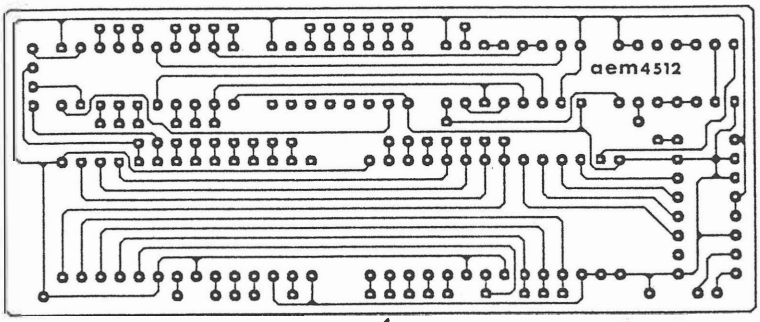
Figure 1. Circuit diagram of the Ultra-graphics adaptor. The 57 inter-board connections, denoted by upper and lower case letters, are detailed in Table 3.

• GA004 in the VZ-300

2 of 8.



Overlay for the printed circuit board showing the placement of components and where the inter-linking wires connect. Note the links on the board. (see Corrections. in Jul 88 p 7. AEm)



Full-size printed circuit artwork. (see Correction in Jun 88 p 7. AEm)

PARTS LIST

Resistors		¼W, 5%
R1,R2		6k8
Semiconductors		
IC1	6264	8Kx8 static RAM
IC2	2764	8Kx8 Char. Set EPROM
IC3	74LS161	
IC4	74LS273	
IC5	74LS138	
IC6	72LS02	
IC7	74LS153	
IC8,		
IC9	74LS244	

Miscellaneous

AEM4512 pc board; 2 x 28-pin low profile IC sockets; thin insulated hook-up wire (ribbon cable).

Price Estimate: $40-$50

* A fully programmable EPROM with the character set in Chart 1 is available from:

Matthew Sorell, 41 Mills St, Clarence Pk, 5034 S.A.

for $18 including postage. Customised character sets are negotiable. Kit suppliers may include pre-programmed EPROMs; check with your supplier first.

such as graphs or high resolution pictures. With an analogue to digital converter, the VZ computer could be easily used as a low cost laboratory computer, able to graph results with acceptable resolution. The highest resolution colour mode (128 x 192 pixels) is also similarly useful.

It is also possible to access the 3 x 2 semigraphics in text mode, which occurs when graphics characters are called while the external character generator is enabled. For more information on the graphics and text capabilities, see the two-part feature "Screen Handling on the VZ200/300", by Bob Kitch, in the September and October 1986 issues of AEM.

CIRCUIT OPERATION

IC5 (74LS138) decodes A4-A7, IORQ and WR to recognise I/O port 20-2FH(32-47). WHen this occurs, pin 15 goes low, causing IC4 (74LS273) to latch the contents of the data-bus (D0-d7). This latch is cleared on RESET to ensure that text is sent to the correct memory page. DA11 and DA12 are bits 00 and 1. They provide bank switching to fit the 8K RAM into the 2K video memory allocation (7000-77FFH (28672 to 30719)). L2, L3 and L4 signals control the graphics mode pins on the 6847 video IC, L5 controls the internal/external character sets and with this the 2 x 2 (normal) or 3 x 2 semigraphics modes. L6 and L7 control whether the inverse and semigraphics modes follow bits 6 and 7 of the character code (normal) or L2 and L3 respectively.

The output of IC1 (6264) controls address lines 4 to 11 of the character EPROM. The EPROM is programmed to mirror the output of IC1 UNLESS the external character set is specifically required. In this case, pin 2 of IC2 is sent high by IC6 (74LS02), which decodes when L5 is high and the video IC is in text mode. IC7 (74LS153) multiplexes the inverse and semigraphics control lines, and is controlled by L6 and L7 to decode L2, L3, D6' and D7'.

IC3 (74LS161) is a synchronous binary counter. It counts through the external character set in the EPROM, so that the correct character row data is released.

Construction

The first thing to do, no matter whether you've purchased a kit or assembled your own parts and made your own printed circuit board, is to check the pc board. See that all the holes are drilled and that there are no broken tracks or tiny copper 'bridges' between the closely-spaced IC pads. Correct any problems you find.

You can commence assembly by first installing the resistors, IC sockets and the non-socketed ICs into the printed circuit board, as shown in the overlay diagram here. The three links should be made on the solder side of the board using insulated wire. Now install the 57 interconnecting wires as required. Make these about 150-200 mm long for the time being. The wire used should be as thin as possible. Separated ribbon cable is quite suitable. The wires should be connect through the component side of the pc board.

Now open the computer by removing the six screws underneath. Remove the main board by undoing the four screws holding it in. Be careful not to flex the keyboard cable too much; if it breaks, it's the devil's own job fixing it. Note which wires go to the power switch and the loudspeaker, then desolder these, leaving the wires on the main pc board.

Desolder the RF shield covering the main board. Use solder wick to do this. Remove the 6116 RAM on the main board, near the TV modulator. The best way to do this is to cut the pins on one side of the IC and wobble it on the other side until the rest of the pins break. Just make sure you've got go the right chip! Now remove the pin stubs left in the pc board. ▷

LEVEL

We expect that constructors of an

INTERMEDIATE

level, between beginners and experienced persons, should be able to successfully complete this project.

3 of 8.

TABLE 1. VZ200 – tracks to cut.

IC	Pin # to	IC	Pin#	Position
6847	29	+5v	-	Adjacent to pin 29 (top side)
6847	32	6847	2	Under 6847 (solder side)
6847	34	6847	40	
6847	40	74LS245	2	Between ICs (solder side)
6847	8	74LS245	3	Between ICs (solder side)
6847	7	74LS245	4	Between ICs (solder side)
6847	6	74LS245	5	Between ICs (solder side)
6847	5	74LS245	6	Between ICs (solder side)
6847	4	74LS245	7	Between ICs (solder side)
6847	3	74LS245	8	Between ICs (solder side)
6847	2	74LS245	9	Between ICs (solder side)
6847	27	Ground		Lift pin out of PCB
6847	30	Ground		Lift pin out of PCB
6847	31	Ground		Lift pin out of PCB

TABLE 2. VZ300 – tracks to cut.

IC	Pin# to	IC	Pin#	Position
6847	32	GA004	27	Under 6847 (solder side)
6847	3	GA004	33	Under 6847 (solder side)
6847	4	GA004	32	Under 6847 (solder side)
6847	5	GA004	31	Under 6847 (solder side)
6847	6	GA004	30	Under 6847 (solder side)
6847	7	GA004	29	Under 6847 (solder side)
6847	8	GA004	28	Under 6847 (solder side)
6847	34	GA004	26	Under 6847 (solder side)
6847	40	6847	34	Under 6847 (solder side)
6847	2	6847	32	Under 6847 (solder side)
6847	27,30,31	Ground	-	Cut, separate and remove track under 6847 (solder side)
6847	29	+5V		Lift pin out of PCB

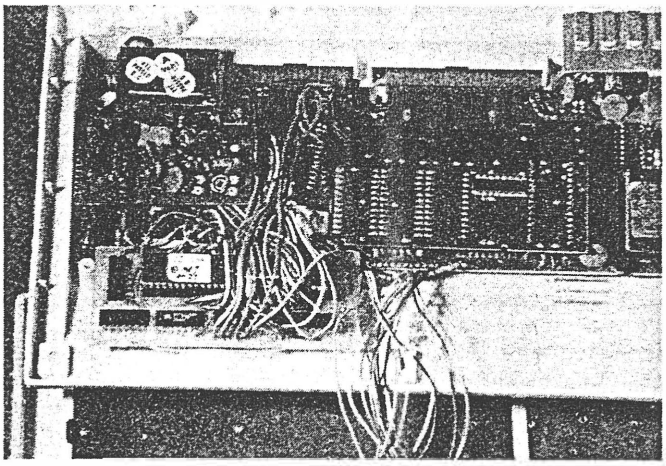


Photo 1. The Ultra-Graphics board installed in the VZ200.

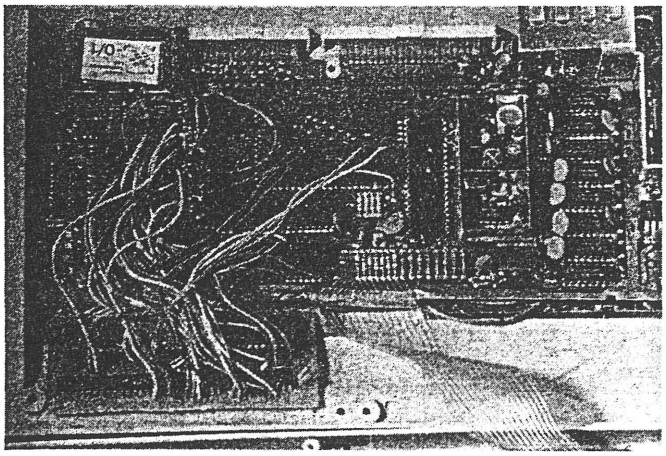


Photo 2. The Ultra-Graphics board installed in the VZ300.

TABLE 3. Interboard connections.

Wire #	VZ-200 IC	Pin #	VZ-300 IC	Pin #
A	6847	36	6847	36
B	6847	38	6847	38
+5V	SUPPLY RAIL		SUPPLY RAIL	
C	74LS245	4	GA004	29
D	74LS245	3	GA004	28
E	74LS245	2	GA004	26
F	74LS245	9	GA004	27
G	6847	40	6847	40
H	6847	2	6847	2
I	6847	8	6847	8
J	6847	7	6847	7
K	6847	6	6847	6
L	(6116)	21	(6116)	21
M	(6116)	23	(6116)	23
N	(6116)	22	(6116)	22
P	(6116)	8	(6116)	8
Q	(6116)	19	(6116)	19
GND	74LS245	10	SUPPLY RAIL	
R	6847	37	6847	37

4 of 8.

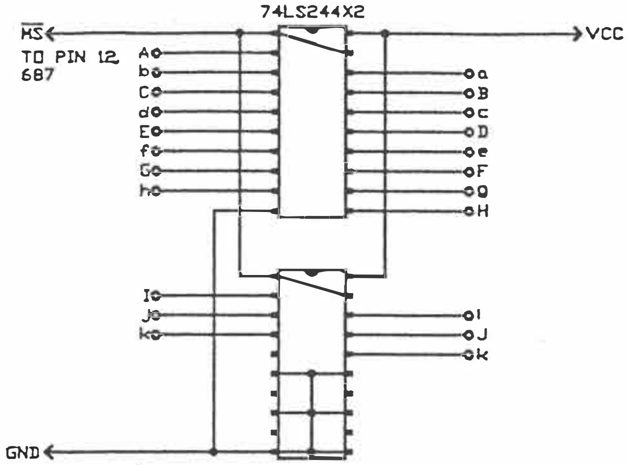


Figure 3. Wiring of the 74LS244 buffers – wrap them in insulation tape once you've got your computer working again.

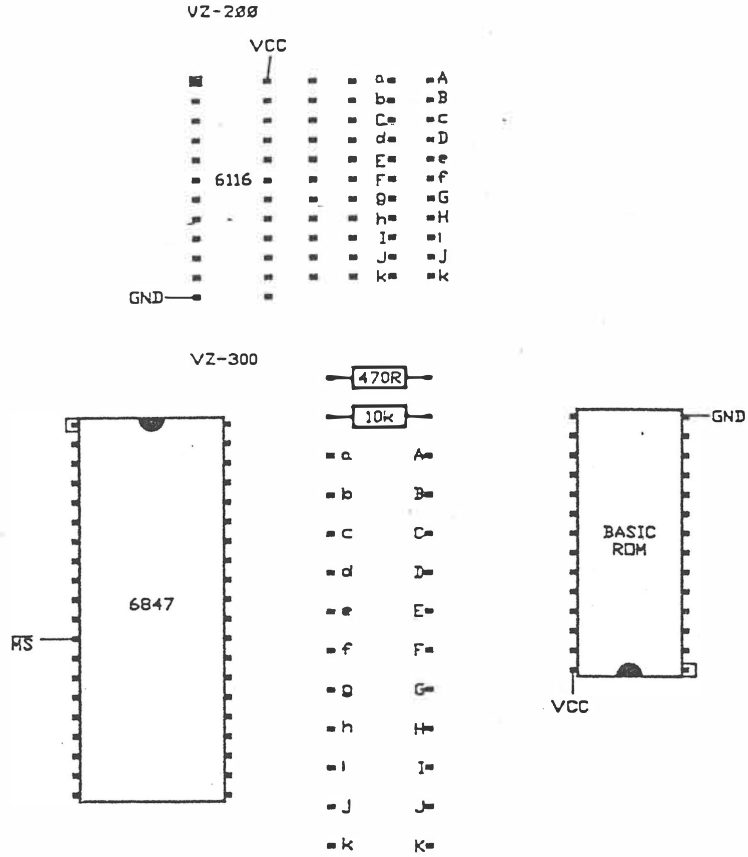


Figure 4. Showing the connection points for the 74LS244 buffers into the VZ200 and 300.

S	74LS245	5	GAØØ4	3Ø	l	74LS245	15	GAØØ4	1Ø
T	74LS245	6	GAØØ4	31	m	74LS245	11	GAØØ4	13
U	74LS245	7	GAØØ4	32	n	6847	27	6847	27
V	74LS245	8	GAØØ4	33	p	74LS245	16	GAØØ4	11
W	6847	3	6847	3	q	74LS245	13	GAØØ4	8
X	6847	4	6847	4	r	6847	32	6847	32
Y	6847	5	6847	5	s	I/O Connector	29	Z8Ø(78ØC)	37
Z	(6116)	7	(6116)	7	t	I/O Connector	12	Z8Ø(78ØC)	36
a	(6115)	1	(6116)	1	u	I/O Connector	27	Z8Ø(78ØC)	34
b	(6116)	2	(6116)	2	v	I/O Connector	5	Z8Ø(78ØC)	2Ø
c	(6116)	3	(6116)	3	w	I/O Connector	14	Z8Ø(78ØC)	22
d	(6116)	4	(6116)	4	x	I/O Connector	1Ø	Z8Ø(78ØC)	35
e	(6116)	5	(6116)	5	y	74LSØ4	4	Z8Ø(78ØC)	26
f	(6116)	6	(6116)	6	z	6847	3Ø	6847	3Ø
g	6847	21	6847	21	1	74LS245	14	GAØØ4	9
h	6847	2Ø	6847	2Ø	2	74LS245	18	GAØØ4	14
i	6847	35	6847	35	3	6847	31	6847	31
j	6847	34	6847	34	4	74LS245	17	GAØØ4	12
k	6847	29	6847	29	5	74LS245	12	GAØØ4	7

5 of 8.

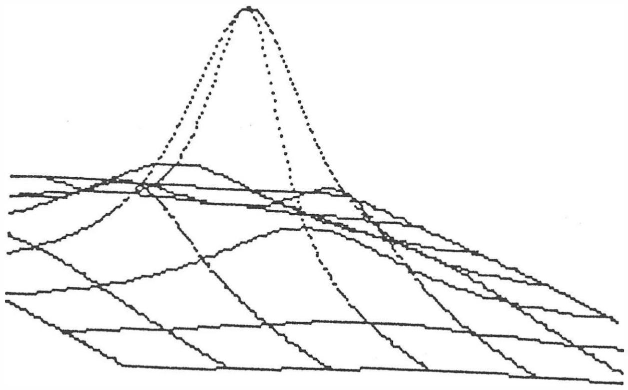


Figure 2. Three-dimensional graphics!

This method greatly reduces overheating problems. Missing tracks are an absolute no-no in computers!

In the VZ200, undo the two screws holding the PAL converter module behind the TV modulator and lift up, to reveal the 6847 video IC. There are two plastic screw mounts on the base of the VZ200. These should be broken off with pliers.

Now the fun begins! Cut the tracks listed in Table 1 (VZ200) or Table 2 (VZ300). Identify each track carefully! Note several IC pins are lifted. When doing this, heat them with a soldering iron and lever the pin out using a small precision screwdriver. Be careful not to break the pin at the IC or all will be lost! Clip off the narrow part of the pin.

Position the Ultra-graphics board in its approximate location relative to the main board. See Photo 1 (VZ200) or Photo 2 (VZ300). Connect each wire in order, as in Table 3, to the main printed circuit board on the *component* side. Cut the wires with a little leeway (about 10 mm longer than required). Tick each connection in Table 3 as it is made, to avoid errors.

Check and recheck all connections. Reconnect the loudspeaker and power switch, fit the main board back into the box (no screws yet) and the new board alongside, as in the photos. Plug in the RAM and EPROM, the video and power supply cables, and switch on. The display should be almost normal. Some characters may be incorrect. The computer should otherwise work correctly. If not, then check for short and open circuits, incorrectly oriented components, and incorrect inter-board wiring.

Unfortunately, the Z80 has trouble controlling the address lines through the resistor buffer with this new board, making the graphics only about 90% accurate. To correct this, power down and then remove the eleven 6k8 resistors on the main board (in the VZ300, do not remove the adjacent 10k and 470R resistors). Wire up IC8 and IC9 (74LS244) as shown in

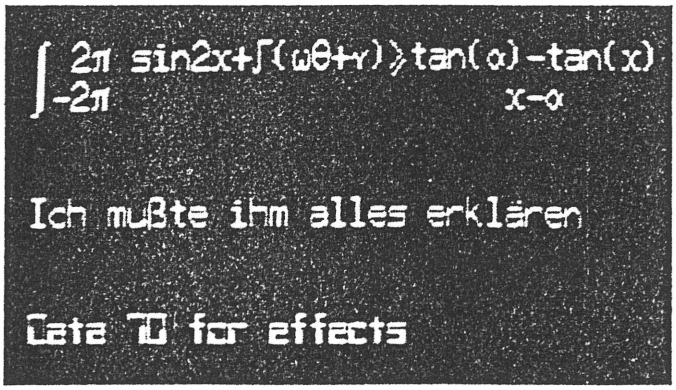


Photo 5. A small taste of what is now possible with text: mathematics, German and Data 70 characters.

Figure 3. Clip the narrow part of each pin, and connect these ICs to the board via short (20 mm) pieces of wire, longer for the power supply and enable signal. Connect them as shown in Figure 4. Wrap these ICs in insulation tape. Switch the computer back on, and when the computer is working, check the new board by typing in:

```
10 CLS:POKE 30744,96:OUT 32.224
20 FOR A=0 TO 255
30 POKE 28672+A,A
40 NEXT
50 PRINT @256,""
```

and running this little program. The new external character set should be displayed.

Screw the board into the box, and the cover on top. Voila. Ultra-Graphics!

The RF shield can be reinstalled, but creates a few problems with mounting the new board. It is not *essential* for the computer's operation and can be left out if you wish.

Applications

It's no use having a set of useful new features without suitable applications with which to exploit them.

The Word Processor

The character set has been designed to be used in conjunction with the tape version of the word processor. Not having used the cartridge version, I don't know how the new character set should be enabled, or if it is compatible with this word processor. To enable a suitable character set, type in:

POKE 30744,96:OUT 32,160

before loading the word processor. Upper and lower case will be enabled, and semigraphics characters will be used as markers. You will find that the word processor is now considerably easier to use.

Text in BASIC

When using the external character set with BASIC, the white-on-black screen should be enabled. BASIC revision 1.2 uses only this mode, but version 2.0 boots up in black-on-white (inverse mode). Since characters 96 to 127 are non-standard, the white-on-black mode should be enabled by typing POKE 30744,96; or by keeping CTRL depressed when turning the computer on.

As mentioned earlier, characters 128-255 can be accessed by poking the correct code onto the screen, or by printing the correct graphics character in the correct colour. This is how photo six was produced. Characters 64 to 127 can be accessed as inverse characters. The character sets available are listed in Table 5.

Using Graphics

The computer now boots in graphics mode 0, so before any commercial software (games) can be loaded, you should type in:

OUT 32,8

to enable the normal graphics mode.

If you have a GP-80 printer, which is compatible with the graphics dump screen, then it is possible to dump games screens by playing the game in graphics mode 6 (128 x 192) on the second RAM page (OUT 32,25). Connect a reset pushbutton to ground on pin 13 (VZ200) or pin 11 (VZ300) of the 74LS04. Reset the computer at a suitable point in the program, and print the screen by typing in:

MODE (1):OUT 32,25: COPY:OUT 32,0

6 of 8.

Using Extension Graphics

The following graphics modes are available:

GM0	OUT 32,0	64x64	Colour	1024 Bytes
GM1	OUT 32,4	128x64	Monochrome	1024 Bytes
GM2	OUT 32,8	128x64	Colour	2048 Bytes
GM3	OUT 32,12	128x96	Monochrome	1536 Bytes
GM4	OUT 32,16	128x96	Colour	3072 Bytes
GM5	OUT 32,20	128x192	Monochrome	3072 Bytes
GM6	OUT 32,24	128x192	Colour	6144 Bytes
GM7	OUT 32,28	256x192	Monochrome	6144 Bytes

The COLOUR command is valid for all colour modes. To set or reset a pixel in each mode, in mode 1, refer to Table 4. To clear the screen in modes 4 to 7, MODE(1) must be enabled on all RAM pages used. This means that the GM7 screen is cleared by using:

OUT 32,30:MODE(1):OUT 32,29:MODE(1):OUT 32,28:MODE(1)

The method is similar for the other modes. A three dimensional plot, based on a Microbee program, but using Graphics Mode 7 instructions is reproduced here.

Listing 1 is a graphics dump routine for Graphics mode 7, written for Shinwa-compatible dot matrix printers, such as the BMC BX-80. The author would appreciate hearing from anyone writing applications software for this graphics modification.

TABLE 4. SET/RESET in graphics modes.

```
GM0:     SET(X+64*(Y AND1),INT(Y/2))
         RESET(X+64*(Y AND1,INT(Y/2))

GM2:     SET(X,Y)
         RESET(X,Y)

GM4:     OUT 32,16+INT(Y/64)AND1:SET(X,Y AND 63)
         OUT 32,16+INT(Y/64)AND1:RESET(X,Y AND 63)

GM6:     OUT 32,24+INT(Y/64)AND3:SET(X,Y AND 63)
         OUT 32,24+INT(Y/64)AND3:RESET(X,Y AND 63)

GM1:     A=28672+INT(X/8)+16*Y
SET:     POKE A,PEEK(A) OR 2^(7 AND (NOT X))
RESET:   POKE A,PEEK(A) AND (NOT 2^(7 AND (NOT X)))

GM3:     Same as GM1

GM5:     OUT 32,20+INT(Y/64)AND1
         Then same as GM1

GM7:     OUT32,28+INT(Y/64)AND3:A=28672+INT(X/8)+32*(YAND63)
         Then same as GM1
```

```
10 REMARKABLE GM7 GRAPHICS DUMP     BY MATTHEW SORELL 17/1/88
20 REM FIND TOP OF MEMORY
30 TM=PEEK(30897)+256*PEEK(30898):TM=TM-201:TL=TM-65536
40 POKE30897,(TL AND 255):POKE30898,TM/256
50 REM PUT PROGRAM AT T.O.M.
60 TM=TM+1:IFTM)32767THENTL=TM-65536 ELSE TL=TM
70 FOR A=TL TO TL+200
80 READ D:POKE A,D:NEXT
90 'CORRECT ABSOLUTE ADDRESSES
100 FORI=1TO20
110 READA,D:POKE TL+A,(TL+D)AND255:POKE TL+A+1,(TM+D)/256:NEXT
150 POKE30862,TL AND255:POKE30863,TM/256
160 REM X=USR(0] STARTS DUMP
170 CLEAR 50:END
180 'MACHINE CODE DATA
190 DATA245,197,229,62,27,205,186,58,62,49,205,186,58,62,13,205
200 DATA186,58,175,50,0,0,198,28,211,32,175,50,0,0,62,13,205
210 DATA186,58,62,27,205,186,58,62,75,205,186,58,175,205,186
220 DATA58,62,2,205,186,58,175,50,0,0,62,7,50,0,0,175,50,0,0
230 DATA175,50,0,0,33,0,112,237,75,0,0,203,56,48,2,203,249,9,58
240 DATA200,192,7,7,7,7,7,79,6,0,9,58,0,0,71,62,1,7,16,253,166
250 DATA40,22,58,0,0,237,68,198,3,7,71,62,3,7,16,253,71,58,0,0
260 DATA128,50,0,0,58,0,0,60,254,4,32,185,58,0,0,205,186,58,205
270 DATA186,58,58,0,0,61,254,255,32,160,58,0,0,60,254,32,32,147
280 DATA58,0,0,60,254,16,194,0,0,58,0,0,60,254,3,194,0,0,6,8
290 DATA62,13,205,186,58,16,251,225,193,241,201,0,0,0,0,0,0
300 'ABSOLUTE ADDRESS CORRECTION DATA
310 DATA20,197,28,196,56,195,61,198,65,199,69,200,76,195,86,200
320 DATA98,198,110,200,125,199,129,199,132,200,140,199,149,198
330 DATA157.195,165,196,171,27,174,197,180,19
```

LISTING 1

TABLE 5. Useful OUT expressions (OUT 32,N).

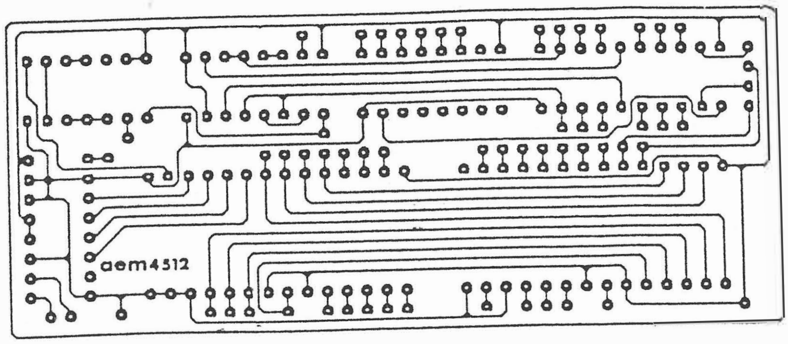
N	GM	Page	Chr 0-63	Chr 64-127	Chr 128-192	Chr 192-255
0	0	0	IntNorm	IntInv	SG4	SG4
4	1	0	IntNorm	IntInv	SG4	SG4
8	2	0	IntNorm	IntInv	SG4	SG4
12	3	0	IntNorm	IntInv	SG4	SG4
16	4	0	IntNorm	IntInv	SG4	SG4
17	4	1	-	-	-	-
20	5	0	IntNorm	IntInv	SG4	SG4
21	5	1	-	-	-	-
24	6	0	IntNorm	IntInv	SG4	SG4
25	6	1	-	-	-	-
26	6	2	-	-	-	-
28	7	0	IntNorm	IntInv	SG4	SG4
29	7	1	-	-	-	-
30	7	2	-	-	-	-
32	0	0	ExtNorm	ExtInv	SG6	SG6
64	0	0	IntNorm	IntInv	IntNorm	IntInv
72	2	0	SG4	SG4	SG4	SG4
96	0	0	ExtNorm	ExtInv	ExtNorm	ExtInv
104	2	0	SG6	SG6	SG6	SG6
128	0	0	IntNorm	IntNorm	SG4	SG4
132	1	0	IntInv	IntInv	SG4	SG4
160	0	0	ExtNorm	ExtNorm	SG6	SG6
164	1	0	ExtInv	ExtInv	SG6	SG6
192	0	0	IntNorm	IntNorm	IntNorm	IntNorm
196	1	0	IntInv	IntInv	IntInv	IntInv
224	0	0	ExtNorm	ExtNorm	ExtNorm	ExtNorm
228	1	0	ExtInv	ExtInv	ExtInv	ExtInv

Int=Internal Chr Inv=Inverted Text Norm=Normal Text
SG4=2x2 Graphics SG6=3x2 Graphics Ext=External Chr

7 of 8.

OOPS! The pc board artwork for the AEM4512 VZ Ultra-Graphics Adaptor was reproduced upside-down with the board number right-reading. Strange?

Here it is, the correct way.



Project 4512, VZ "Ultra-Graphics Adapter", April '88. On the overlay, 'V' goes to pin 10 of IC2 (the 2764) and 'j' is missing – it goes to a pad just above pin 9 of IC7 (the LS153), presently obscured by the point of the V. On the circuit (p.58), IC3 (the LS161) has pins 3, 6, 7, 10 and 16 shown earthed when they go to +5 V, while pins 4,5 and 8 were omitted – they go to earth.

8 of 8.

Better VZ amp

Anyone who tried to build the VZ published in the May 1988 edition of this magazine may have had a few problems with it. Here are some modifications.

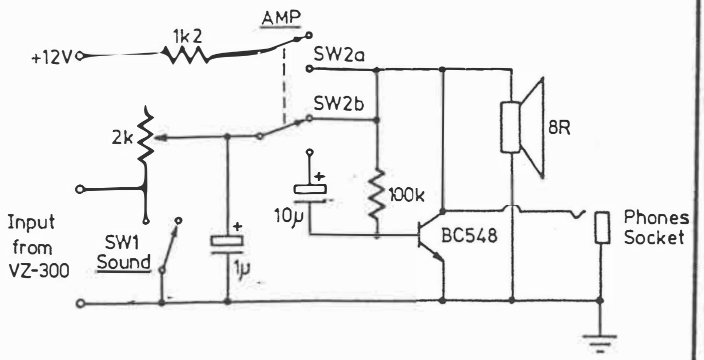
Shorting out the speaker is not very healthy for the computer as it either causes the computer to crash or the program to go haywire. The remedy is to put the switch inline with the speaker.

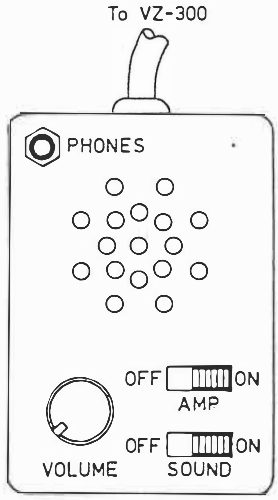
The volume is not very loud so I reduced the 1K2 resistor to 12OR. The volume control acted more like a tone control so I re-connected it (see circuit diagram).

An on/off switch is not needed if you take the positive power supply from the internal switch.

I didn't use a 6.5mm plug and socket to connect up the amp and computer, mainly because I had a 3.5mm plug and socket but also I didn't want to remove the monitor socket, so I put the socket on the top left hand side of the computer near the vent.

Ben Hobson,
Quirindi,
NSW 2343.





VZ Amp

One of the main downfalls of the VZ300 is its inferior sound, which is brought about by the inefficiency of the piezo-electric speaker included in the unit.

It is for this reason that I have designed a circuit to replace the piezo speaker with a magnetic one and include other features such as an amplifier and volume controls.

The volume and sound on/off controls run directly from the supply used for the piezo but the amplifier needs a 12 V supply which is taken from the input socket for the transformer. I cut the wires from the piezo and removed the monitor socket (because I'm using the TV socket), and installed a 6.5 mm stereo socket in its place. This socket must be stereo because it has to handle the two connections from the piezo and the supply rail. It is essential to check with a multimeter for the polarity of the speaker before cutting it and connecting it to the socket.

I used figure-eight shielded cable to connect the computer to the enhancer mainly because of the three individual connections. The sound can be turned off when you are doing a lot of typing, eg word processing, so you are not annoyed by constant beeping. Headphones can be used when there are other people in the vicinity that do not want to be disturbed.

Switch 1 (sound) is used to turn the sound on/off which it does by either creating a short circuit in parallel to the speaker or opening this short circuit. Switch 2 (amp) is a double pole to switch both the amplifier circuit and to turn on the power to this section.

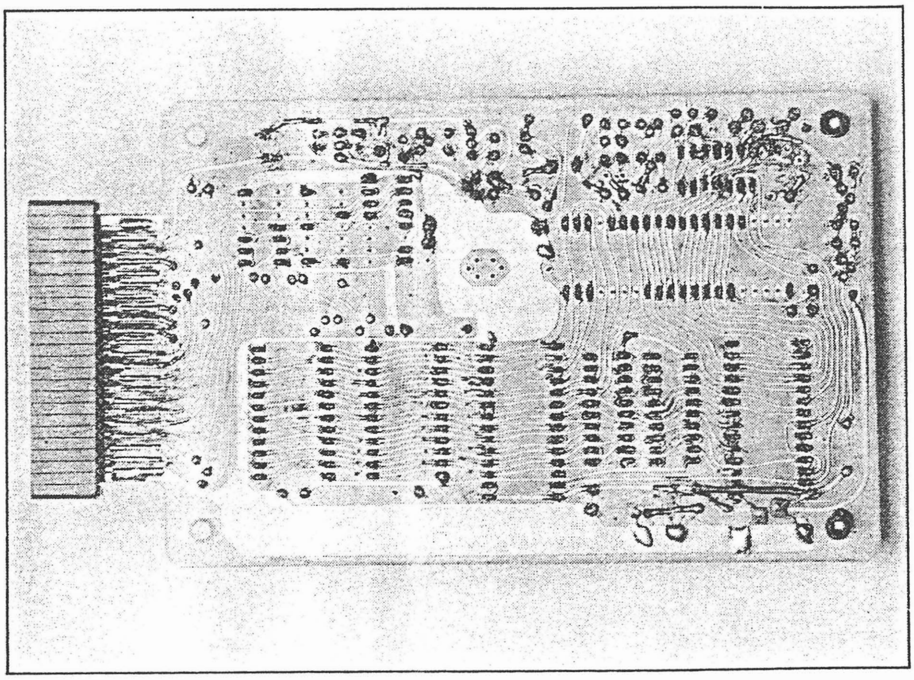
Steven Merrifield
Newlyn North Vic

ETI-1611: VZ300 EPROM programmer

Part 1

Customise your computer with this EPROM programmer. This month the hardware, next month, the software.

Herman Nacinovich



FOR ANYONE SERIOUSLY involved with microprocessors or computers, this EPROM programmer will prove to be an invaluable tool. It has lots of features, some of which may only be found in commercial programmers costing much more. Yet it uses relatively few parts, including cheap, readily available IC's and discrete components. Everything is on a single board which plugs directly, or via a ribbon cable plus socket, into the memory expansion slot of a VZ300 computer. Power for the programmer is derived from the internal power supply of the VZ300, thereby saving the cost of having a separate power supply. Also, there is no need for a housing and this represents a further saving in cost.

I designed this EPROM programmer for use with a VZ300 computer for the simple reason that I happen to have a VZ300. Apart from that, however, the choice of a VZ300 for this application has the advantage that it is available at a very attractive price, yet it is more than adequate for the job. In fact, the total cost of this EPROM programmer plus a VZ300 may be less than the cost of a commercial programmer with similar features but without the computer. Thus, if you need an EPROM programmer but don't have a VZ300, it might be worth considering whether the low cost of this computer would justify its purchase for this application. After all, a second computer can always come in handy, can't it?

Among the features built into the EPROM programmer is versatility. This is because most of its operation is under software control. This includes selection of programming voltages appropriate to EPROMs from different manufacturers, modes of data transfer and editing capabilities. There are no switches as these are made unnecessary by virtue of the software programmability.

A ZIF (zero insertion force) socket is provided on the board for a 28-pin EPROM to be programmed. There is provision on the board for an optional, second ZIF socket for a second EPROM which has already been programmed. This allows direct copying from one EPROM to another. In addition, there is provision for an optional 4K of RAM which can be used to extend the internal RAM capacity of the VZ300. This can be useful for editing or for temporarily storing large chunks of machine code before burning them into an EPROM. Also, with 4K of RAM, the board can be used to extend the memory capacity of the VZ300 when it is not used to program EPROMs.

With suitable software, this EPROM programmer can be programmed to do such useful things as verify whether an EPROM has been fully erased before programming, copy from one EPROM to another (as mentioned), transfer data from EPROM to RAM and vice versa, manually enter data temporarily into RAM and editing before transferring to EPROM. One of the good features of this EPROM programmer is that the software can be modified to extend its capabilities without any changes to the board.

The programmer is designed primarily for programming 28-pin EPROMs of the 2764, 27128 and 27256 types (and their CMOS equivalents). There are, of course, other types around, but to try to cater for

1 of 5.

all available types would require a horrendously complex switching arrangement and an overall cost which could not be justified. Besides, many of the earlier types (such as the 2708) would seem to be obsolete, hard to get and, on top of that, ridiculously expensive. On the other hand, the 2764, 27128 and 27256 EPROM types would seem to be the most popular and useful currently available. Furthermore, they are substantially pin compatible with each other which simplifies the design of a programmer considerably. With these points in mind, it seems reasonable to limit the design of a programmer for use with these three EPROM types as a compromise between versatility and circuit complexity.

EPROM Characteristics

For those not fully familiar with EPROM characteristics, a general description of these devices may be useful.

All EPROMs of the types with which we are concerned have a set of Address pins, a set of DATA pins and a set of CONTROL pins. The number of address pins reflects the bit capacity of an EPROM. Thus, the 2764 (64K bits) has 13 address pins, the 27128 (128K bits) has 14 address pins and the 27256 (256K bits) has 15 address pins. All EPROMs of this series have eight data pins. That is, data bits are programmed into, and read out of, these devices as 8-bit groups, or bytes.

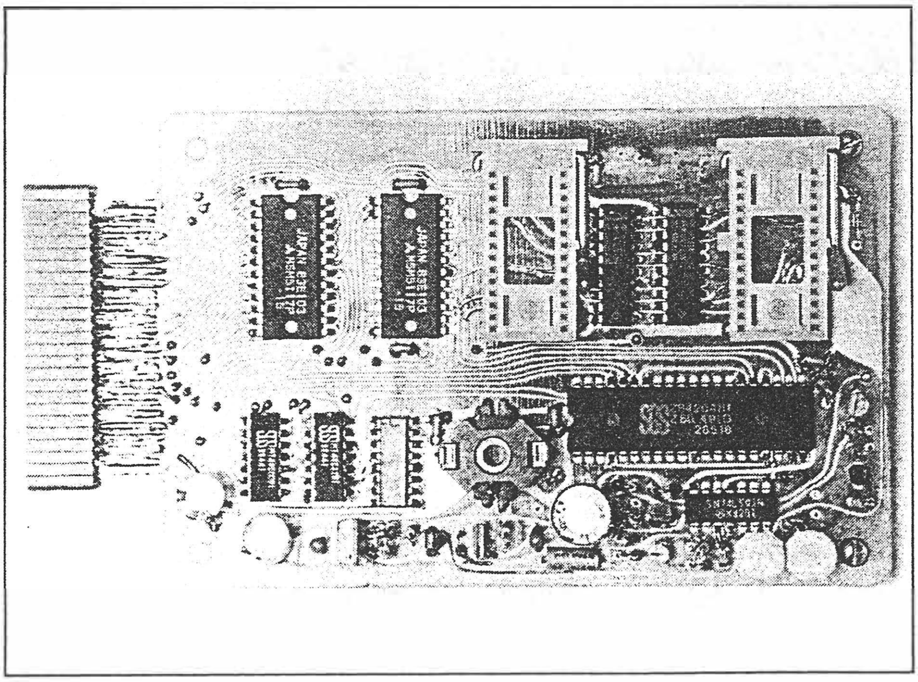
The control pin functions are labelled CE (chip enable), OE (output enable) and PGM (program). The bars over these letters mean that these functions are activated by a logic LOW signal and, conversely, de-activated by a logic HIGH signal, at the respective pins. In the 27256, the CE and PGM functions are combined and accessed at a single pin, while in the other two EPROM types these are associated with separate pins. Incidently, all address and control signals are specified to be at TTL levels.

In addition to ADDRESS and DATA pins, these EPROMs have a GROUND (0V supply), Vcc (+5V supply) and Vpp (programming voltage supply). Vpp is specified to be +5V for READ operations and either +12.5V or +21V (typically), depending upon the manufacturer, for PROGRAMMING operations.

In a READ operation, an address is sent to the address pins and OE and CE are brought LOW. The byte stored at that address in the EPROM appears at the DATA pins and is read. During all read operations, Vpp must be kept at +5V.

A PROGRAM operation is more complicated. Vpp is raised to a high voltage level as specified by the manufacturer. An address is sent to the address pins while a byte to be programmed into that address location is sent to the data pins. CE and PGM are brought momentarily LOW. The usual practice is then to verify that the eight data bits have been correctly programmed before proceeding to program data into the next address location. In the verify operation, the address and Vpp are maintained in their previous states, while OE is brought LOW. The programmed data bits appear at the data pins and are read. If the bits are verified as being correctly programmed then programming proceeds to the next address.

During programming, only 0's can be programmed into selected bit locations. It is not possible to reverse the process by electrically changing a 0 bit to a 1 bit. Thus, initially, all bits in an unprogrammed EPROM must be at a logic 1 and that is generally the case with all EPROMs as they come from the manufacturer. If, for any reason, some of the bits are at logic 0 before programming, then the entire EPROM will have to be erased by exposure to UV radiation. An EPROM programmer, therefore, should be capable of verifying, before programming, that an EPROM has been fully erased. As implied, erasure is the process of converting



ETI-1611 — PARTS LIST

Resistors — all resistors ¼W5% tolerance unless stated otherwise.

R1 .... 68R
R2, 3, 9, 10, 11, 13 5K6
R3, 9 .... 560R
R4, 5 .... 3K3
R6, 7 .... 820R
R8 .... 560R
R12 470
R14 .... 2K2
R15 .... 15K
R16 .... 47R
R17 .... 150R
R18 .... 5K6
R19 .... 5K6
R20 .... 1K
RV1, RV2 .... 1K

Semiconductors

IC1, IC2 .... 74HC373
IC3 .... 74 LS 151
IC4 .... Z80A PIO
IC5 .... 74L500
IC6 .... 74L532
IC7 .... 74LS05
IC8 7805
RAM1, 2 .... 6116, optional
Q1 .... BD139
Q2, 3, 4, 5 .... BC557
Q6 .... BC547
D1, 2, 4 .... 1N4148

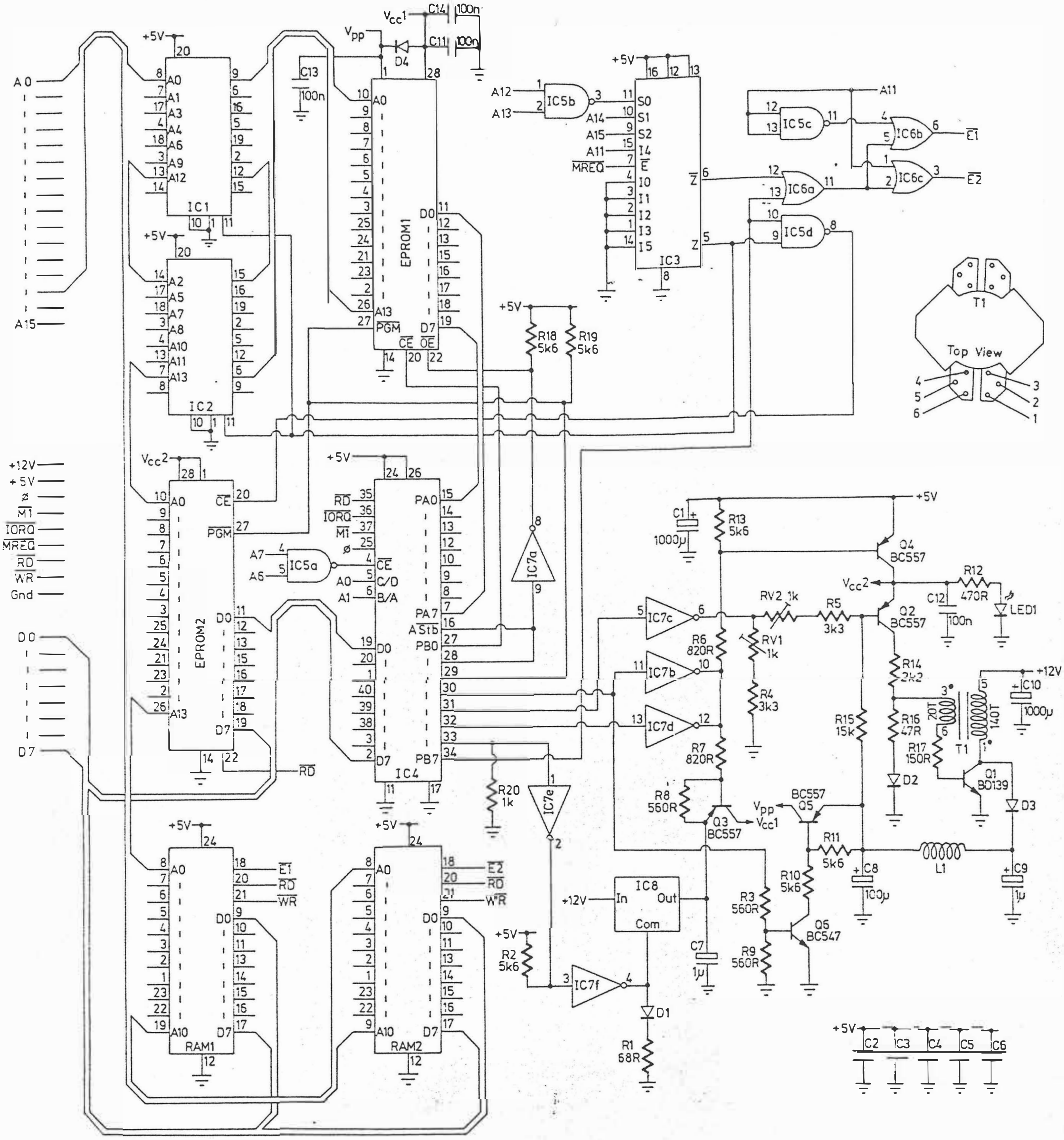
Capacitors

C1, C10 .... 100µ/16V electrolytic
C2-C6, 11, 12 .... 100n monolithic ceramic
C7, 9 .... 1µ tag tantalum
C8 .... 100µ/25V electrolytic (330µ/50v used in prototype)
C13, 14 .... 100n monolothic ceramic

Miscellaneous

PC board, double sided; ZIF socket — Part No P 0655 (Altronics); ZIF socket, optional; inductor, 56µH; ferrite pot core, Neoside, Part No 29-813-25; former to suit, Neosid, Part No 60-793-64; clips to suit, Neosid, Part No 76-022-95; Edge connector, approx. 6m of 0.25mm diam. enamelled copper wire, pair rubber feet, 10mm long, 1 pair of screws, 3mm x 6mm, with nuts.

2 of 5.



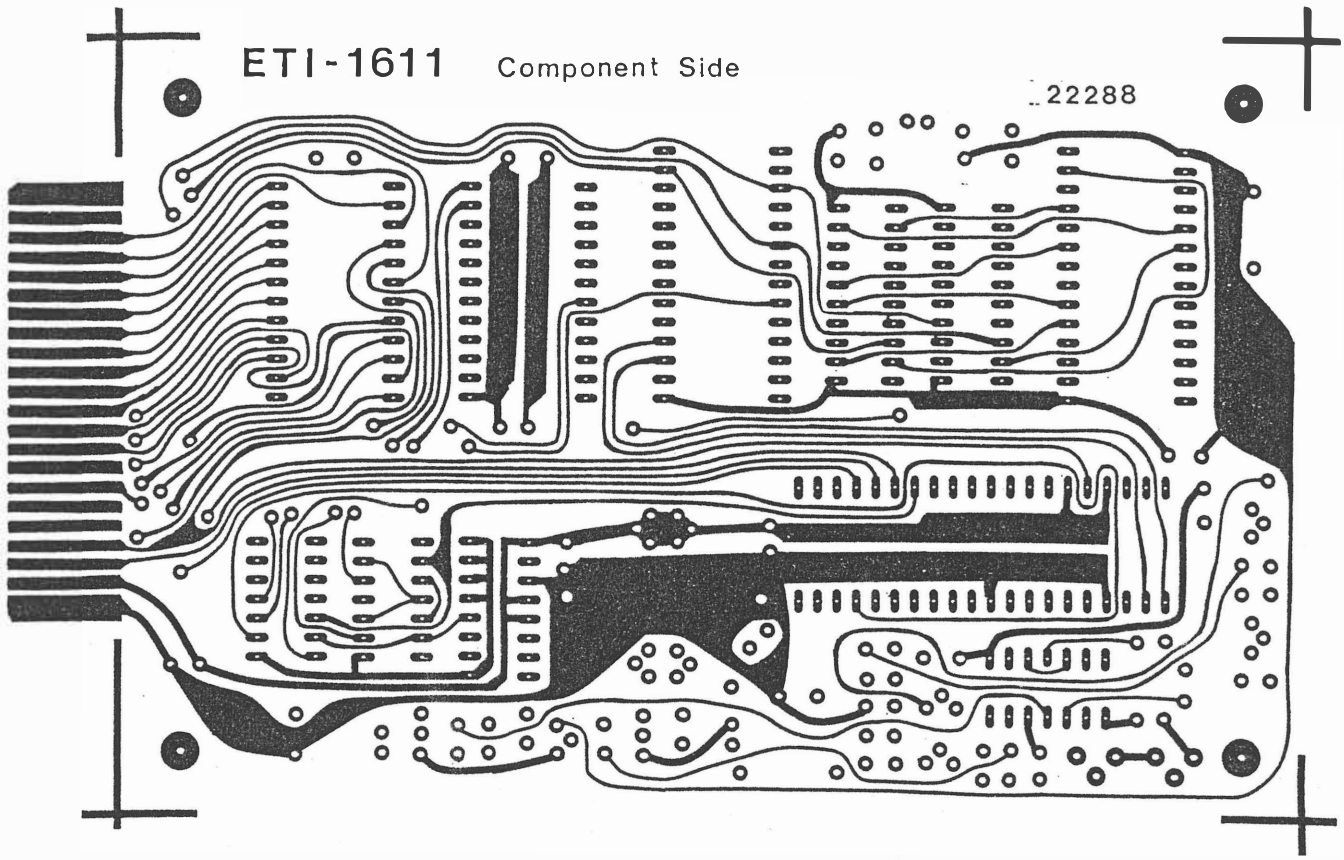
ETI-1611 — HOW IT WORKS

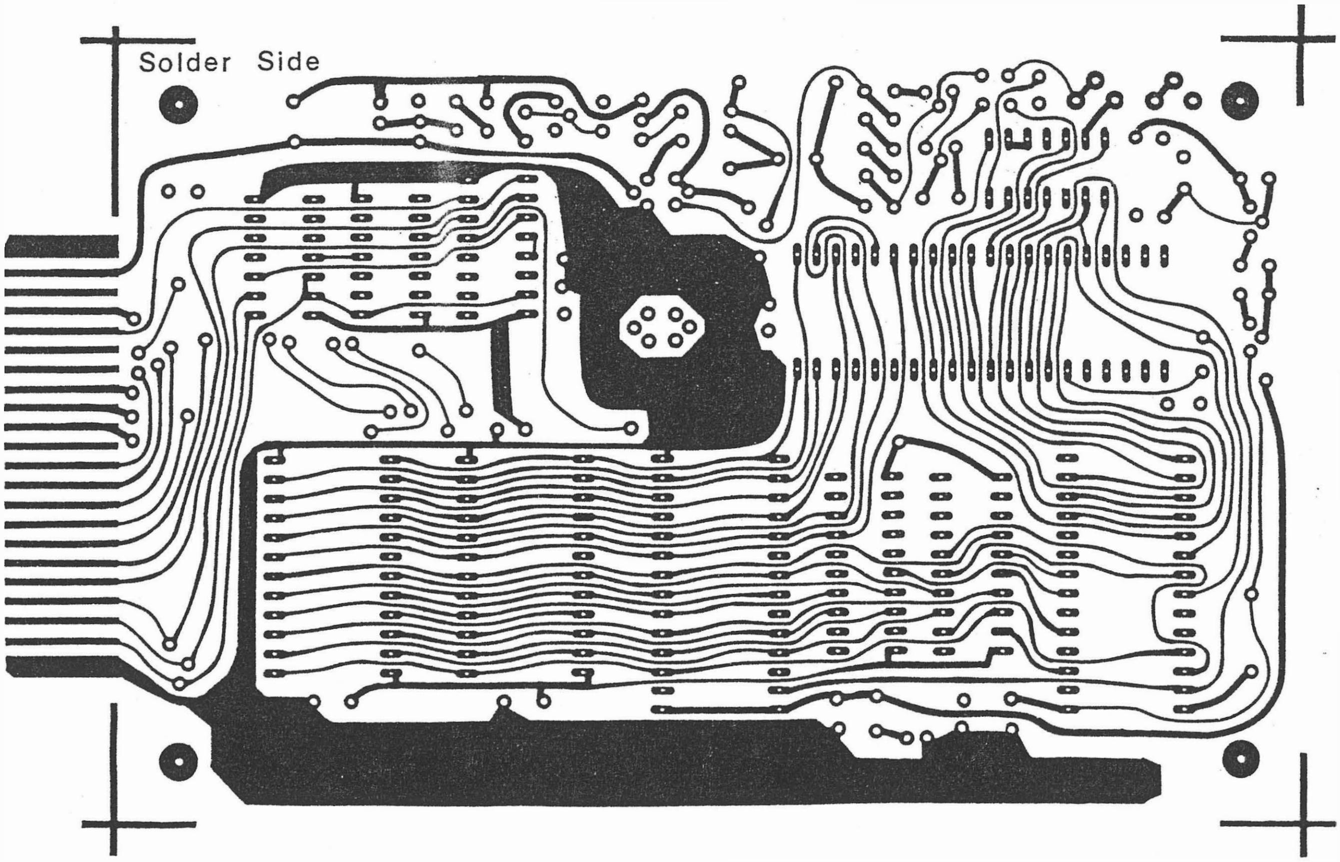
As it happens, the VZ300 has unused memory address space in the range B800H to FFFFH, which is available for external memory expansion, etc. Address decoder IC3 generates enable signals for the address latch, on-board RAM and EPROM 2 whenever the VZ300 executes a memory read or write instruction for an address in this range. When IC1 and IC2 have been enabled, the address is latched in their outputs and sent to the address inputs of EPROM 1.

IC4 provides the interface between the VZ300's microprocessor and EPROM 1 and the associated control circuitry. In use, PORT A is programmed by instructions from the VZ300 for bidirectional data transfer between the VZS300 and EPROM 1. PORT B is programmed as an output port, also by instructions from the VZ300, and generates all the necessary control signals for EPROM read and program operations in response to an appropriately coded instruction from the VZ300. During an EPROM read operation, data is read by an IN instruction addressed to PORT A. During an EPROM program instruction, data is sent to PORT A by an OUT instruction addressed to that port.

RAM 1 and RAM 2 share a common address range with EPROM 2. To avoid conflict, the decode circuitry allows only one of these to be enabled at any one time. Whether the EPROM or one of the RAMs is selected depends on a control bit sent to port B.

The total address space available for external memory in the case of the VZ300 is only a little over 16K. To program a 27256, which has 32K bytes capacity, it is necessary to generate the most significant address bit by some means other than via the VZ300's address bus. The problem is solved by using one of the port B lines for this purpose. As it happens, the PGM CONTROL Pin on the 2764 and 27128 becomes the most significant address pin on the 27256, so the same port B line is used to control both functions. The only complication is that slightly different software is needed for the 27256.





4 of 5.

all the bits in the EPROM to a logic 1 by exposure of the EPROM chip to UV radiation. For this purpose, EPROMs are provided with a transparent quartz window above the chip. This window should be covered by an opaque label to prevent accidental erasure in the case of a programmed EPROM. Not all EPROMs, however, are erasable (despite the name). The exception is known as a 'one-time-programmable EPROM', which is an ordinary EPROM but without the quartz window. This device is fully erased when leaving the factory and can only be programmed once. It is intended for use in production equipment and has the advantage of being cheaper to make than an erasable EPROM because a quartz window is not required.

It appears that most problems encountered by EPROM users arise due to faulty or incomplete programming. A marginally programmed bit, for example, may verify OK immediately after programming but may subsequently revert to the opposite logic level while the EPROM is in service. To guard against this possibility, National Semiconductor recommend, for their CMOS range of EPROMs, that programming and verification be carried out with Vcc raised to 6V and that Vcc be lowered to the normal 5V level for ordinary read operations. It seems that, with Vcc raised to 6V, a marginally programmed bit will verify as being unprogrammed, whereas the same bit may not do so with Vcc at 5V. Raising Vcc to 6V during programming and verification guarantees that all bits verified as being correctly programmed will read correctly during service. It will be noted, however, that 6V exceeds the 5.5V maximum operating level generally specified for EPROMs and manufacturers' specification should always be consulted if in doubt. In any case, the present EPROM programmer can be programmed to apply either 5V or 6V to Vcc during programming according to the user's selection.

An important consideration, also, when programming EPROMs, is the width of the PGM pulse which is applied during programming. Older EPROM types such as the 2708 were specified to be programmed with a single 50mS pulse per address location. With many later types, typified by the 27064 to 270256 series, a maximum pulse width as short as 10mS may be specified. Some manufacturers recommend an interactive programming algorithm to minimise the overall programming time. In an example of such an algorithm, a programming pulse of 0.5mS is applied and the programmed byte is verified. If it verifies as correctly programmed then programming proceeds to the next address. If not, then another 0.5mS pulse is applied with the current address and the process repeated until the byte verifies OK. If, after 20 pulses, a given address still does not verify OK then the EPROM is rejected as unprogrammable. With the present programmer it is a simple matter to adapt the software to any programming algorithm that may be recommended by an EPROM manufacturer.

Circuit Description

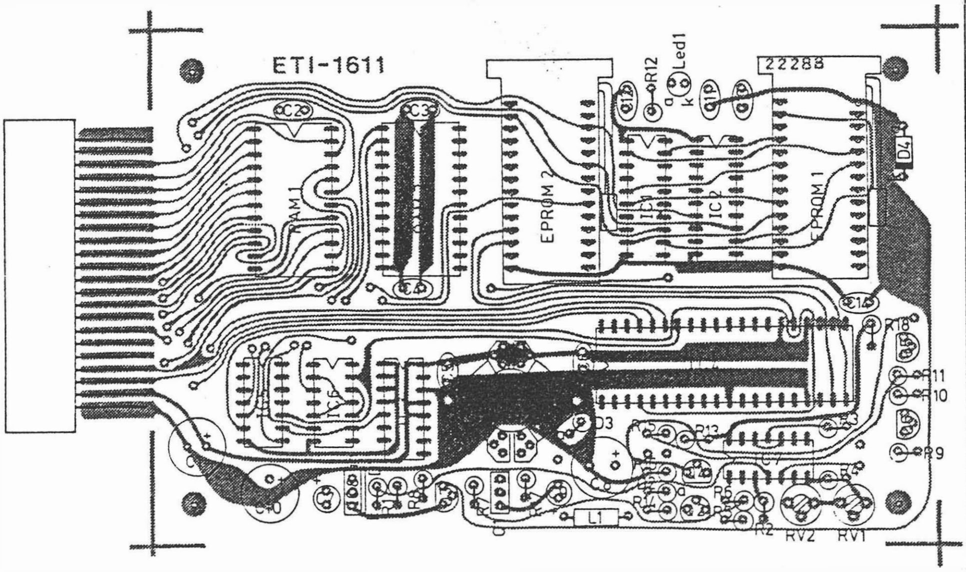
When plugged into the memory expansion slot of a VZ300 computer, this EPROM programmer has direct access to the address, data and control lines of the VZ300's internal Z80 microprocessor. Additionally, the memory expansion bus provides a 5V regulated supply voltage and a 12V unregulated supply voltage. There are 16 address lines and 8 data lines. The main control lines are MREQ (memory request), IORQ (input/output request), RD (read), WR (write) and O (clock).

The circuit comprises two 8-bit registers (IC1 and IC2) wired as a 14-bit address latch. IC3 and IC5b form an address decoder and IC4 provides a programmable interface between the VZ300's microprocessor and EPROM 1 which is the EPROM to be prrogrammed. A 28-ZIF socket is provided on the board to enable the EPROM to be easily inserted and removed. Although more expensive than an ordinary IC socket, this type saves a lot of frustration and effort and is well worth the cost. There is space on the board for an optional, second, ZIF socket for EPROM 2. This is provided in case there is a need to copy from one EPROM to another as quickly as possible. Data can be programmed into, or read from EPROM 1 but can only be read from EPROM 2.

There is also space on the board for an optional pair of 2K static RAMs (RAM 1 and RAM 2). This allows for up to 4K of extra RAM if desired. As previously noted, this can be useful for temporarily storing large chunks of machine code and also allows the board to be used as a handy 4K expansion board for a VZ300 when it is not used for programming EPROMs.

The high Vpp voltage required for programming is generated on the board by a fly-back type DC-DC inverter. This comprises a ferrite core transformer T1 and transistor Q1 in a conventional self-oscillating configuration. The Vpp voltage is regulated by Q2, with one of two voltage levels (21V and 12.5V) selected under software control. Transistors Q3, Q4 and Q5 are used to switch off the Vcc and Vpp supply voltages at the respective pins of EPROM 1 and EPROM 2 before an EPROM is inserted into or removed from its socket. Power ON to the EPROMs is indicated by LED 1 lighting up.

The Vcc supply voltage (Vcc1) for EPROM 1 is obtained from a 5V voltage regulator IC (IC8) on the board. Although the nominal output voltage of this IC is 5V, a resistor R1 and diode D1 connected in series from the 'COM' terminal or IC8 boost the output voltage to around 6V (plus or minus 0.25V). This higher than normal voltage for Vcc is available when programming an EPROM (subject to recommendations of the EPROM manufacturer) and is reduced under software control to 5V in the EPROM read mode. Vcc supply voltage (Vcc2) for EPROM 2 is derived from the VZ300's 5V supply. ●



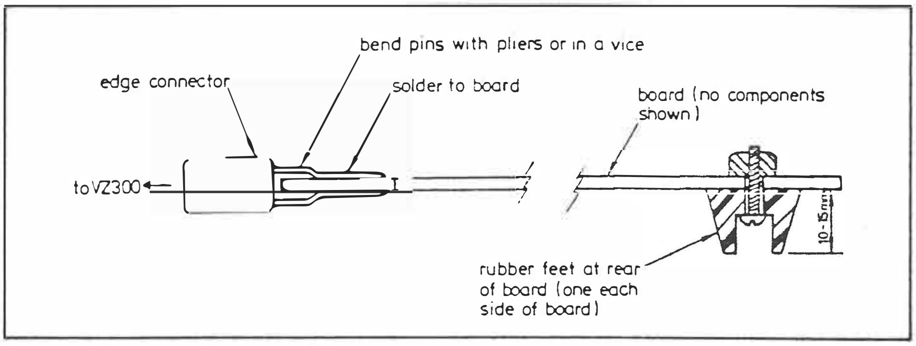
5 of 5.

ETI-1611 EPROM programmer

This month part 2 continues with construction, testing and software for the programmer.

Herman Nacinovich

COULD YOU USE a low cost EPROM programmer that will program EPROMs in the popular 2764 to 27256 series? How about one that will also copy from one EPROM to another in seconds? And one which is fully software programmable to cater for EPROMs from different manufacturers, with different programming voltages? How about an EPROM programmer which can double as a memory expansion for a VZ3000 computer when it is not being used for programming EPROMs? Or, one which can load your favourite BASIC program directly from an EPROM into your VZ300 computer in a matter of seconds? This is it!



The bond plugs directly into the VZ300 expansion stat. The foot at right releives mechanical stress on the connector at left.

Construction

Construction of this project is simplified by the fact that everything goes on a single board and there is no messing about with wires, switches and a box to put everything into. However the usual, if not more, care, should be exercised to ensure that everything is put in the right way around, particularly the IC's, diodes, transistors and electrolytic capacitors. The board uses double sided construction and boards as supplied by kit suppliers will (hopefully) have plated through holes. Although the number of parts on the board is not too great, there are lots of tracks on the board and many of these are very closely spaced. It is strongly recommended that the greatest care be taken in the first instance when examining the board to ensure that there are no bridges or breaks in the board pattern. Never assume that any board, whether you make it yourself or get it in a kit, is free of faults.

It is also important to be careful, when winding the ferrite core transformer, to ensure that the ends of the windings are connected the right way around. If not, the inverter won't work and you might find transistor Q1 getting very hot. The particular ferrite core and former recommended here are made by Neosid, the core and former being very kindly supplied for the prototype by Neosid Limited Australia in Lilyfield, NSW. It is a very easy transformer to wind and the former is of moulded construction with integral pin terminals, virtually guaranteeing success providing that reasonable care is taken in putting the transformer together. One point to watch, however, is that the pin terminals are fairly small and close together. This is no real problem but a steady hand and a pair of long-nosed pliers with very thin, pointed ends do help when trying to twist the wire ends around the pins. When that is done, solder the wire ends to the pins and then fit the transformer onto the board.

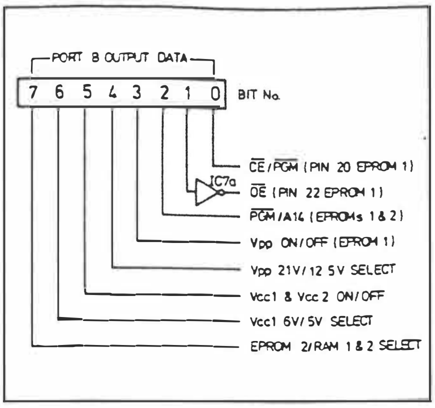
A little hint: It is very difficult to scrape the enamel off enamelled winding wire as required when making solder connections. I have found that by burning the wire end with a lighted match and then rubbing lightly with steel-wool, the enamel comes off very easily.

The board is designed so that it can be plugged into a 44-way edge connector, if desired, with a 44-wire ribbon cable going to a second edge connector which plugs into the memory expansion slot of a VZ300. Alternatively, an edge connector of the type with rearwardly extending solder pins can be soldered directly to the tracks on the edge of the board. This avoids the need for a ribbon cable and was the method chosen for the prototype.

One drawback, however, is that you may not be able to plug in a printer or disc drive at the same time because the memory expansion and I/O slots in the back of a VBZ300 are a little too close together. If you anticipate that you will need to plug in a printer or disc drive at the same time as the EPROM programmer then I would recommend that you use a pair of 44-way edge connectors joined by a ribbon cable.

Apart from the ZIF sockets for EPROMs 1 & 2, IC sockets were not used in the prototype. One reason for this was that the prototype board, though of double sided construction, did not have plated through holes and that would have made soldering IC sockets to the board a little tricky (though not impossible). In the case of the ZIF sockets, I could not solder the pins to both sides of the board as required by the double sided board construction. However, I solved the problem by drilling the IC pin holes on the board a little oversize and linking both sides of each solder

1 of 4



Port B connections

pad, where required, with very fine wire. I would expect boards supplied by kit suppliers to have plated through holes so that, if you choose to use IC sockets, you will find soldering them no more difficult than you would if the board were single sided. One note of advice, however, whether you use IC sockets or not: once you solder an IC or an IC socket, the tracks running underneath the IC or IC socket, on the component side of the board, will no longer be accessible. So, double check for breaks in, or shorts between, tracks underneath each IC before soldering.

On the board there are a number of through-the-board links. Assuming plated-through board construction, these links will be automatically formed in the board as supplied so that there will be need to solder anything to them.

You will notice that there are a couple of trimpots on the board. It may happen that the trimpots supplied with a kit may not quite fit the holes on the board since trimpots come in different sizes with different pin spacings. If you find this to be the case then simply bend the leads (very carefulloy) so that they will fit. Be particularly careful if the trimpots supplied have a ceramic rather than plastic base because the ceramic base is extremely brittle and therefore easily broken.

One last hint: I had a little problem with the locking lever on the ZIF socket being awkward to get at. I solved the problem by bending the lever upwards about a quarter way from its end with a pair of pliers.

Addendum

When you have completed soldering in the components on the board as per the payout given last month, you will have to solder additional resistors R19 and R20 to the reverse side of the board. Solder resistor R19 to pins 24 and 29 of IC4 (Z80 APIO) and resistor R20 directly to pins 1 and 7 of IC7 (74LS05). note: These resistors are not shown in the parts layout published last month.

Testing

Before plugging the board into your VZ300, make one final check over the entire board with a magnifying glass to ensure that there are no breaks in any of the tracks and that there are no solder bridges between tracks. It would also do no harm to check that all the components, particularly IC's diodes, transistors and electrolytics have been soldered in the right way around.

When satisfied that all is OK, plug the board into the VZ300. Do not plug in any EPROM yet. Switch on your monitor and allow it to warm up. Then switch on the VZ300 and observe the display on the screen. If it is the normal display that you get after switch-on then all, so far, is OK. If, however, you get garbage on the screen, or nothing at all, then switch off immediately. In this case there will almost certainly be a fault on the board, either a faulty component or a short circuit between tracks, or a broken track or you may have forgotten to solder one of the components.

Assuming that the display is OK, connect the negative lead of a multimeter (set to read 25 V or more) to a convenient point on the board at 0 V and the positive lead to either end of inductor L1. You should get a reading of around 21 V plus or minus 3 V. If you don't then switch off immediately. Check whether transistor Q1 is hot or cold. If it is hot then most likely one of the windings of transformer T1 has its ends wrongly connected or transposed. If Q1 is cold then possibly the transformer windings are connected to the wrong pins or there is an open circuit somewhere, depriving Q1 of base or collector current. In any case, check the circuit around Q1 and T1 before switching on power again.

Assuming, again, that so far everything is OK the LED should be alright. now, you can check a few voltages on the pins of the EPROM sockets. If these pins are not readily accessible by your multimeter probes, you could plug in a conventional 28-pin IC socket into each ZIF socket. Most ordinary IC sockets have the pin connectors exposed, making access with a probe easy.

At this stage, with the VZ300 switched on, you should be able to measure 5V (plus or minus 0.25V) at pins 1,27 and 28 of both EPROM 1 and EPROM 2 sockets.

Now key in the following: OUT203,7:OUT203,15:OUT201,7:OUT201,143. The LED should light up and you should get the following:

EPROM 1 socket: pin 1: 21 V plus or minus 3V, pin 27: 5 V, pin 20:4 V, pin 22: 0V, pin 28: 6V plus or minus 0.25 V. Adjust RV2 for 21 V at pin 1. Now key in: OUT202,40. You should now get the following voltages at the pins of EPROM 1: pin 1: 12.5 V plus or minus 2 V, pins 20,27: 0V, pin 22: 5 V, pin 28, 5 V. Adjust RV1 for 12.5 V at pin 1.

Now key in: OUT 202,2. The LED should now go out and you should get a reading of 0V at each of pins 1, 20,22,27 and 28.

This completes the preliminary testing of the EPROM programmer and also demonstrates how the various operating voltages and control functions for the EPROMs are software controlled.

The ultimate testing is carried out by loading a suitable program into the VZ300 and running that program with an EPROM plugged in. To save you, dear reader, the rather time consuming task of writing a program, such a program has been developed for you. Unfortunately, this has turned out to be a somewhat lengthy one and space limitations prevent publication here. However, the program is available on EPROM from the author at the address given at the end of this article. Loading the program into the VZ300 involves merely plugging the EPROM into the EPROM 2 socket on the programmer board and keying in a few short instructions in BASIC. The actual loading takes only a few seconds, compared with loading the same program from tape, which would take many minutes. This program will enable you to manually enter machine code into memory, make corrections, if necessary, and then copy into EPROM. The program will also let you check if an EPROM is fully erased and you can copy from one EPROM to another. EPROMs from 2764 to 27256 are catered for.

I would like to express my particular appreciation of Neosid Australia for their assistance. I had a lot of difficulty trying to find a suitable ferrite core from other sources for this project. Neosid came to

Bit No. →	7	6	5	4	3	2	1	0
8-bit word to PORT B →	0	0	1	0	0	1	1	0

How the bits are arranged on Part B

2 of 4

the rescue with just the right core for the job.

SOFTWARE DESCRIPTION

This programmer board is configured to appear partly as an external memory and partly as an I/O (INPUT/OUTPUT) device to the Z80 microprocessor of a VZ300 computer. There are, therefore, four primary instructions in BASIC which are needed to communicat with and control the programmer. These are:

POKE (write into memory)
OUT (write into I/O)
PEEK (read from memory)
INP (read from I/O)

The VZ300 computer has 16K of internal ROM (Read Only Memory), occupying addresses 0000H (Hex) to 3FFFH and 16K of user accessible RAM (Random Access Memory) occuypying addresses from 7800H to B7FFH. The space occupying addresses B800H to FFFFH (a little over 16K) is vacant in the VZ300 and is available for memory expansion. In addition, the Z80 microprocessor in the VZ300 is capable of addressing up to 256 bytes in I/O space, independently of memory, by an I/O instruction.

The part of the programmer board which appears as external memory comprises up to 4K of optional RAM (RAMs 1 & 2) and up to 32K of optional ROM (EPROM 2). These share the same address space but only one of these can be selected at any one time. Whichever is selected depends on the status of one of the bits of an 8-bit control word contained in an I/O instruction.

In addition, address latches IC1 and IC2 are enabled by any memory read or write instruction to the same address space as occupied by the on-board RAM and ROM. Being 'write only', these latches do not pose any danger of bus conflict but it will be noted that any address stored in the outputs of these latches will be changed to a new address whenever a read or write instruction is sent to either of the external RAMs or ROM. The latched outputs are unaffected by any instruction to an address below B800H. When an address is latched into IC's 1 & 2, this becomes the address for EPROM 1 (the EPROM which is to be programmed).

IC4 provides the I/O interface between the I/O part of the EPROM programmer and the VZ300's internal Z80 microprocessor. As already noted, the Z80 is capable of addressing up to 256 bytes in I/O space (from 00H to FFH). A simple address decoder (IC5a) enables IC4 for I/O instructions to addresses C0H to FFH (192 decimal to 255 decimal). This leaves addresses 00H to VFH availble for other devices (disc, printer, etc) which may be plugged into the I/O expansion slot, next to the memory expansion slot, of a VZ300. IC4 has two 8-bit I/O ports (PORT A and PORT B) which are programmable as either input ports, output ports or as (PORT A only) a bi-directional data transfer between the Z80 microprocessor of a VZ300 and the data pins of EPROM 1. PORT B is used for generating various operating voltages and control signals for EPROMs 1 and 2 and is therefore programmed, in use, as an outport port. Programming of these ports consists in sending the following instructions (in BASIC) after power is switched on and prior to using the EPROM programmer:

OUT 203,7; OUT 203,15: OUT 201.7: OUT 201,143.

Note that the order in which these instructions are sent is important. These instructions are necessary to initialise ports A and B. Once the ports are initialised, data may be written into, or read from either port by appropriately addressed OUT and INP instructions, as follows:

INP 202 — read data from PORT B
OUT 200,A — write data (A) to PORT A
INP (200) — read data from PORT A
OUT 202,B — write data (B) to PORT B

Although PORT B data can be read by an IMP instruction, this instruction is not used for PORT B as it is an output port only in this application.

When either PORT A or PORT B is configured as an output port, data, in the form of an 8-bit word addressed to that port by an OUT instruction, will be latched in an internal register for that port in IC4. At the same time, the data will appear at the I/O pins associated with that port and remain there until a new instruction is addressed to that port.

Each bit of an 8-bit word written into PORT B determines a particular operating voltage or control function associated with the operation of the EPROM programmer. The respective bit allocations are shown in the accompanying diagram.

By way of example: Suppose that we want to set up the following conditions: enable RAMs 1 & 2, set Vcc1 to 5V, switch ON power (Vcc1 & Vcc2) to EPROMS 1 & 2, set Vpp to 12.5 V, switch Vpp OFF, set PGM HIGH, bring OE (EPROM 1) LOW, and bring CE (EPROM 1) LOW.

In this case, the required word, in binary form, which we would write into PORT B would look as follows:

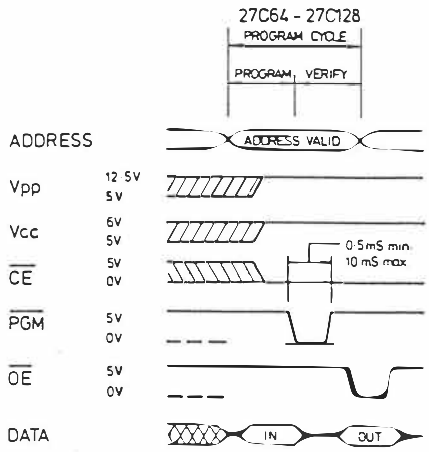
This word corresponds to 26 (hex) or 38 (decimal). The PORT B address is CA (hex) or 202 (decimal). Therefore, to set up PORT B as above, we simply execute the following instruction (in BASIC):

OUT202,38

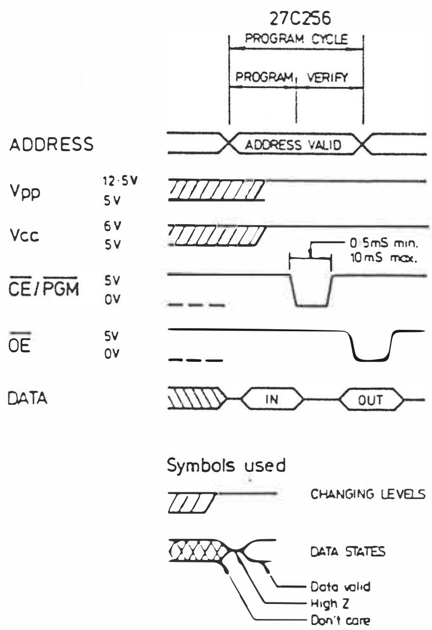
Bit 1, PORT B, determines whether PORT A is an output or an input. When bit 1 is LOW (logic 0), ASTB (pin 16, IC4) is also LOW and PORT A is an output. At the same time OE (pin 22, EPROM 1) is HIGH (logic 1) and any EPROM plugged into the EPROM 1 socket will have its output buffers disabled. That is, data can be written into EPROM 1 via PORT A, as will be the case during an EPROM programming cycle.

When bit 1, PORT B, is HIGH, ASTB goes HIGH and OE goes LOW. Data can now be read from EPROM 1 via PORT A.

An EPROM programming cycle is an operation in which a specified programming voltage (Vpp) and a programming



Waveforms. At top for 2764/128 type devices, at bottom for 256.



3 of 4

pulse (PGM) of specified duration, together with a desired address and data are applied to the appropriate pins of an EPROM. It is the job of the software to generate the necessary voltages and signals in accordance with the manufacturer's specifications for the EPROM to be programmed. Figure 4 show typical wafeforms for 27C64 to 27C256 type EPROMs as recommended by National Semiconductor.

EPROM manufacturers generally recommend programming algorithms designed to give maximum programming efficiency and speeds when programming their EPROMs. The software developed for this project uses a programming algorithm adapted from one recommended by NATIONAL SEMICONDUCTOR for their CMOS range of EPROMs. This programming algorithm has been found to work well with NMOS EPROMs from other manufacturers. For those interested, a flowchart of the programming algorithm used by the software developed for this project is given in Fig. 5.

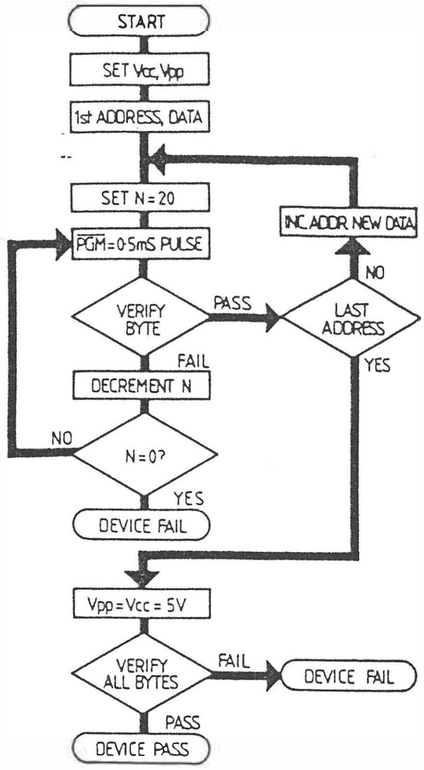
It is anticipated that most constructors in this project will take advantage of the software offer made in this article. Space does not permit a full description of the features which this software offers although a brief outline has already been given. A lot of effort has been put into its development to make this EPROM programmer project versatile, easy to use and as foolproof as possible.

If, however, you choose to develop your own software for this programmer, be careful to ensure that it will generate the correct voltage levels and programming (PGM) pulse widths in accordance with the various EPROM manufacturers' specifications. Some hints for you: write any subroutine for generating the EPROM programming pulses in machine language (for Z80 microprocessors) and ensure that the sub-routine starts with an 'interrupt disable'. If possible, check out all waveforms on a CRO before trying out the programmer on an EPROM.

Whether you purchase the software or write your own, be careful to observe the following precautions:

1. Never insert or remove an EPROM from a socket while power is ON (as indicated by the LED lighting up.

2. Always ensure that the correct programming voltage (Vpp) is selected before programming an EPROM. Different manufacturers specify different programming voltages. The following is a list of EPROM types vs manufacturer and programming voltages derived from information given in the 1987 JAYCAR catalogue and reproduced here with their kind permission (note that ETI cannot accept any responsibility for any errors which may occur in this list) ●



Software flow chart

Eproms can be ordered from the author at Beryl Road, Gulgong, NSW 2852.

EPROM/Manufacturer		programming voltage
2764	Intel, Fairchild, OKI, NEC, TI, Toshiba, AMD, Fujitsu, Hitachi	21V
2764A	Intel, AMD	12.5V
27C64	National	12.5V
27128	Intel, AMD, Fujitsu, NEC, Toshiba, TI, Mitsubishi (M5L27128K & M5M27C128K)	21V
27128	ADC, AMD, NMC27CP128, NMC27C128C (National)	12.5V
27256	Intel, Atmel, NMC27C256 National	12.5V
TMM27256D	Toshiba, Fujitsu (27256 & 27C256)	21V

If in any doubt, always check with the manufacturer.

4 of 4.

```
2 POKE30897,255:POKE30898,174:CLEAR100:POKE30744,1
3 OUT203,7:OUT201,7:OUT203,15:OUT201,143:OUT202,0
4 N=-20736
6 READX
8 IFX<0THENRESTORE:GOTO40
10 POKEN,X
12 N=N+1:GOTO6
20 DATA243,33,0,0,17,0,0,1,0,0,237,176,201,243
22 DATA33,0,0,17,0,0,1,0,0,237,184,201,0,0,0,0,0,0
24 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
25 DATA243,33,000,000,17,000,000,221,33,000,000,221,126,0,211,200
26 DATA126,62,0,211,202,14,20,62,0,211,202
27 DATA6,95,58,239,104,203,103
28 DATA200,16,254,62,000,211,202,62,000,211,202,34,056,175
29 DATA219,200,221,70,0,144,62,0,211,202,40,10,126,13,32,215
30 DATA62,0,50,058,175,201,237,82,200,0,25,35,221,35,24,187
32 DATA33,000,000,17,000,000,221,33,000,000,62,000,211,202,126
33 DATA58,239,104,203,103,200,62,000,211,202,219,200,221,70,0
34 DATA144,34,056,175,62,000,211,202,40,6,62,0,50,058,175,201
35 DATA237,82,200,25,35,221,35,24,215,0,0,0,0
37 DATA243,253,229,221,33,0,0,253,33,0,0,17,1,0,33,0,0
38 DATA62,164,211,202,221,070,0,0,062,32,211,202,0,0,0,253,112,0
39 DATA221,035,253,35,167,237,82,032,229,253,225,201,-1
40 CLS:PRINT"ETI 1611 EPROM PROGRAMMER","--------------------";
42 PRINT"----------","KEY     OPERATION","---  --------------
44 PRINT" 1   LIST/ENTER PROGRAM"," 2   VERIFY ERASED EPROM
46 PRINT" 3   COPY RAM TO EPROM "," 4   COPY EPROM 2 TO EPROM 1
48 PRINT" 5   SPECIFY EPROM TYPE"," 6   DEC. TO HEX. CONV.
50 PRINT" 7   HEX. TO DEC. CONV."," 8   COPY EPROM 2 TO RAM"
51 PRINT
52 SOUND15,5:INPUT"WHEN READY ENTER SELECTED KEY";K$:SOUND15,1
54 IFK$="1"THEN190
56 IFK$="2"THEN700
58 IFK$="3"THEN900
60 IFK$="4"THEN1100
62 IFK$="5"THENGOSUB800:GOTO4
64 IFK$="6"THEN750
66 IFK$="7"THEN780
68 IFK$="8"THEN500
70 IFK$="9"THEN100
80 PRINT"INCORRECT KEY ENTRY. TRY AGAIN":SOUND10,4:GOTO40
190 CLS:PRINT"LIST ENTER DATA (HEX)",
192 PRINT"-------------------------------"
195 PRINT:PRINT"ADDRESS       OP-CODE":A$="":A=0
200 PRINTTAB(1);RIGHT$("0000"+A$,4),
205 D$=A$:D=A
210 INPUTA$:L=LEN(A$)
212 IFA$=""THEN1300
213 IFA$="-"THEN600
214 IFA$="+"THEN650
215 IFA$="END"THEN40
220 IFL=2ORL=4GOSUB300:GOTO230
221 SOUND10,3
225 PRINT"ILLEGAL CODE ERROR, TRY AGAIN":A$=D$:A=D:GOTO200
230 IFFLAG=1THENFLAG=0:GOTO225
235 IFL<>2THEN260
240 POKE-18432+D,A:IFPEEK(-18432+D)=ATHEN280ELSE265
260 IFA>4095THEN265
262 PRINTCHR$(27);"                         "
263 PRINTCHR$(27);:GOTO200
265 SOUND10,3
270 INPUT"OUT OF MEMORY RANGE,DO YOU WANT TO CONTINUE Y/N";K$
275 IFK$="Y"THENA$="0":D$=A$:A=0:D=0:GOTO200ELSE20
280 IFD>4095THEN265
282 PRINT,CHR$(27);A$;"   "
285 A=D+1:IFA>4095THEN265ELSEGOSUB400:GOTO200
300 N=0:M=1/16:B=0:A=0
305 L=LEN(A$):IFL=0THEN320ELSEB$=MID$(A$,L-N,1):FLAG=0
310 IFASC(B$)>47ANDASC(B$)<58THENB=ASC(B$)-48:GOTO325
315 IFASC(B$)>64ANDASC(B$)<71THENB=ASC(B$)-55:GOTO325
```

1 of 5.

```
320 FLAG=1:RETURN
325 A=A+16*M*B:M=M*16:N=N+1
330 IFN=LTHENRETURNELSE305
400 C=A:A$="":REMDEC TO HEX SUB
405 B=C-16*INT(C/16):C=INT(C/16)
410 IFB<10THENB$=RIGHT$(STR$(B),1):GOTO420
415 B$=CHR$(B+55)
420 A$=B$+A$
425 IFC>0THEN405ELSERETURN
500 CLS:E0=E1AND3:IFE0=0THENGOSUB800:E0=E1AND3:CLS
502 PRINT"COPY EPROM 2 INTO RAM","--------------------------------
504 PRINT"DO NOT INSERT OR REMOVE EPROM   WHILE LED IS ON",,,,
506 PRINT"EPROM TYPE",
508 IFE0=1PRINT"2764/27C64":MF= 8191
510 IFE0=2PRINT"27128/27C128":MF=16383
512 IFE0=3PRINT"27256/27C256":MF=32767
514 PRINT:INPUT"IS EPROM TYPE CORRECT Y/N";K$:PRINT
516 IFK$<>"Y"THENGOSUB800:E0=E1AND3:CLS:GOTO0502
518 INPUT"ENTER EPROM START ADDRESS (HEX)";K$:IFK$="END"THEN4
520 A$=K$:GOSUB300:IFFLAG=0THEN528
522 PRINT"ILLEGAL CODE ERROR. TRY AGAIN":SOUND10,9
524 PRINTCHR$(27);"                                ";CHR$(27);
526 PRINT CHR$(27);"              ":PRINTCHR$(27);CHR$(27);:GOTO0518
528 IFA=<MFTHENMS=A:GOTO0538
530 PRINT"OUT OF EPROM ADDRESS RANGE.     TRY AGAIN":SOUND10,9
532 PRINTCHR$(27);"                ",,CHR$(27);CHR$(27);"            ";
534 PRINT"                             ",CHR$(27);CHR$(27);"
536 PRINTCHR$(27);"                        ",CHR$(27);CHR$(27);:GOTO0518
538 OUT202,164:CNT=2:S=160:FORN=1TO500:NEXT
540 B=A+4095:IFB>MFTHENB=MF:BYTE=B-A+1ELSEBYTE=B-A+1
541 IFA>16383THENA=A-16384:CNT=1:S=164
542 IFB>16383THENB1=B-16384:B=16383ELSECNT=1
544 IFE0<>3THENOUT202,164:CNT=1:S=164
546 M1=49152+A:M2=47104:M3=B-A+1
548 H=INT(M1/256):L=M1-256*H:POKE-20532,L:POKE-20531,H
550 H=INT(M2/256):L=M2-H*256:POKE-20528,L:POKE-20527,H
552 H=INT(M3/256):L=M3-H*256:POKE-20522,L:POKE-20521,H
554 POKE-20519,S
556 POKE30862,199:POKE30863,175
557 X=USR(X):IFCNT=1THEN560
558 CNT=1:S=164:M2=47105+B-A:M1=49152:M3=B1:GOTO0548
560 PRINT "DATA TRANSFER TO RAM COMPLETE",,,"NO OF BYTES";
562 A=BYTE:PRINTA;:GOSUB400:PRINT"  (";A$;"--HEX-)":OUT202,0
564 SOUND15,5:PRINT:INPUT"PRESS [RETURN] WHEN READY";K$:GOTO04
600 PRINT,CHR$(27);"DATA DELETED ":M=47104+D:N=4096-D
610 HI=INT(M/256):LO=M-256*HI:POKE-20730,HI:POKE-20731,LO:M=M+1
620 HI=INT(M/256):LO=M-256*HI:POKE-20733,HI:POKE-20734,LO
630 HI=INT(N/256):LO=N-256*HI:POKE-20727,HI:POKE-20728,LO
640 POKE30862,0:POKE30863,175:X=USR(X):A$=D$:GOTO0200
650 PRINT,CHR$(27);"INSERT DATA":M=51199:N=4096-D
660 HI=INT(M/256):LO=M-256*HI:POKE-20720,HI:POKE-20721,LO:M=M+1
670 HI=INT(M/256):LO=M-256*HI:POKE-20717,HI:POKE-20718,LO
680 HI=INT(N/256):LO=N-256*HI:POKE-20714,HI:POKE-20715,LO
690 POKE30862,13:POKE30863,175:X=USR(X):A$=D$:GOTO0200
700 OUT202,0:E0=E1AND3:IFE0=0THENGOSUB800:E0=E1AND3
702 CLS:PRINT"THIS SUBROUTINE VERIFIES IF      EPROM ERASED"
704 PRINT"--------------------------------":COUNT=0
705 PRINT"DO NOT INSERT OR REMOVE EPROM   WHILE LED IS ON",,,,
706 PRINT:PRINT"EPROM TYPE",
708 IFE0=1THENPRINT"2764/27C64":M2=57343:P2=0
710 IFE0=2THENPRINT"27128/27C128":M2=65535:P2=0
712 IFE0=3THENPRINT"27256/27C256":M2=65535:P2=1
714 IFE0>3THEN700
716 PRINT:INPUT"IS EPROM TYPE CORRECT Y/N";K$
718 IFK$="END"THEN4
720 IFK$<>"Y"ANDK$<>"YES"THENGOSUB800:E0=E1AND3:GOTO0702
```

2 of 5

```
721 OUT202,37:FORX=0TO1000:NEXT
722 IFE0=3THENS1=33OR(4*P2):S2=34OR(4*P2)
724 IFE0<>3THENS1=37:S2=38
726 POKE-20678,1:POKE-20595,0:POKE-20594,192:POKE-20592,255
727 POKE-20591,M2/256:POKE-20585,S1:POKE-20574,S2:POKE-20569,6
728 POKE-20568,255:POKE-20567,0:POKE-20561,S1
729 POKE30862,140:POKE30863,175:X=USR(X):COUNT=COUNT+1
730 OUT202,0:IFPEEK(-20678)=0THEN738
732 P2=P2-1:IFP2=0THEN722ELSESOUND15,5
734 SOUND10,5:PRINT,,,"EPROM ERASE VERIFY OK":PRINT
736 PRINT:INPUT"DO YOU WISH TO DO AGAIN Y/N";K$
737 IFK$="Y"THEN700ELSE4
738 PRINT:PRINT:SOUND10,9:PRINT"EPROM NOT FULLY ERASED"
740 PRINT:PRINT"NO OF BYTES CHECKED:";
742 PRINTPEEK(-20680)+256*PEEK(-20679)-65536+16384*(COUNT)+1
744 PRINT:INPUT"DO YOU WISH TO DO AGAIN Y/N";K$
746 IFK$="Y"THEN700ELSE4
750 CLS:PRINT"THIS SUBROUTINE WILL CONVERT A "
752 PRINT"DECIMAL INPUT TO ITS HEX EQUIV."
754 PRINT"-------------------------------":PRINT
756 PRINT"WHEN FINISHED ENTER [END]":PRINT:PRINT
758 PRINT"INPUT DECIMAL","HEX EQUIV.","-------------","---------
760 INPUTK$:IFK$="END"THEN20ELSEA=VAL(K$)
762 PRINT,CHR$(27);:GOSUB400:PRINT"     "+A$:GOTO760
780 CLS:PRINT"THIS SUBROUTINE WILL CONVERT A  HEX INPUT TO ";
782 PRINT"ITS DEC EQUIV"
784 PRINT"--------------------------------":PRINT
786 PRINT"WHEN FINISHED ENTER [END]":PRINT:PRINT"INPUT HEX",
788 PRINT"DECIMAL EQUIV.","----------","-------------
790 INPUTA$:IFA$="END"THEN40
792 GOSUB300:IFFLAG=0THEN796
794 FLAG=0:PRINT"ILLEGAL CODE ENTRY. TRY AGAIN":SOUND10,5
795 GOTO790
796 PRINT,CHR$(27);A:GOTO790
800 CLS:PRINT"SELECT EPROM TYPE","----------------------------";
806 PRINT"-----",,"  TYPE","KEY","  ----","---","2764/27C64"," 1
814 PRINT"27128/27C128"," 2","27256/27C256"," 3",,,
820 INPUT"PLEASE ENTER SELECTED KEY";K$:IFK$="END"THEN20
822 IFK$="1"ORK$="2"ORK$="3"THENE1=VAL(K$):GOTO830
824 PRINT:PRINT"WRONG KEY ENTRY. TRY AGAIN":SOUND10,9:GOTO800
828 PRINTCHR$(27);:PRINTCHR$(27);:PRINTCHR$(27);:GOTO820
830 CLS:PRINT"PLEASE SPECIFY PROGRAMMING"
831 PRINT"VOLTAGE 21V/12.5V"
832 INPUTK$:IFK$="21"ORK$="21V"THENE1=E1+16:GOTO838
833 IFK$="END"THEN4
834 IFK$="12"ORK$="12V"ORK$="12.5"ORK$="12.5V"THEN838
835 IFK$="END"THEN4
836 PRINT@256,"WRONG ENTRY. TRY AGAIN":SOUND10,9:GOTO830
838 PRINT:PRINT"PLEASE SPECIFY VCC DURING"
840 PRINT"PROGRAMMING, WHETHER 6/5V"
842 INPUTK$:IFK$="6"ORK$="6V"THENE1=E1+64:GOTO854
844 IFK$="5"ORK$="5V"THEN854
845 IFK$="END"THEN4
846 PRINT:PRINT"WRONG ENTRY. TRY AGAIN":SOUND10,9
848 PRINTCHR$(27);"                                    "
850 PRINTCHR$(27);:PRINTCHR$(27);:PRINTCHR$(27);"                    "
852 PRINTCHR$(27);:GOTO842
854 RETURN
900 OUT202,0:E=E1AND3:IFE=0THENGOSUB800
901 CLS:PRINT"COPY RAM TO EPROM 1":PRINT"----------------------";
902 PRINT"-----------":PRINT"DO NOT INSERT OR REMOVE EPROM"
903 PRINT"WHILE LED IS ON",,,,"EPROM TYPE",
904 E0=E1AND3:IFE0=1THENPRINT"     2764/27C64":BYTE=8192:GOTO912
906 IFE0=2THENPRINT"  27128/27C128":BYTE=16384:GOTO912
908 IFE0=3THENPRINT"  27256/27C256":BYTE=32768:GOTO912
912 PRINT"PROGRAMMING VOLTAGE      ";
914 E=E1AND16:IFE=0THENPRINT"12.5V"ELSEPRINT"21V"
916 PRINT"VCC DURING PROGRAMMING    ";
```

3 of 5

```
918 E=E1AND64:IFE=0THENPRINT"5V"ELSEPRINT"6V"
922 PRINT:INPUT"IS ABOVE INFORMATION CORRECT Y/N?";K$:FLAG=0
923 IFK$="END"THEN4
924 IFK$<>"Y"GOSUB800:GOTO900
926 INPUT"ENTER START ADDRESS IN EPROM    (HEX)";A$:GOSUB300
927 IFA$="END"THEN40
928 IFFLAG=0THEN938
930 PRINT"ILLEGAL CODE ENTRY. TRY AGAIN":SOUND10,9
932 PRINTCHR$(27);"                                         ";CHR$(27);
934 PRINTCHR$(27);"                                         ";CHR$(27);
936 PRINT:PRINTCHR$(27);:GOTO926
938 IFA<BYTETHEN948
940 PRINT"OUT OF EPROM ADDRESS RANGE      TRY AGAIN"
942 SOUND10,9:PRINTCHR$(27);"                                         "
944 PRINTCHR$(27);"                               "
946 PRINTCHR$(27);:GOTO932
948 IFA<16384THENM1=A+12*4096:BIT2=0:GOTO952
950 M1=A+12*4096-16384:BIT2=1
952 PRINT:INPUT"HOW MANY BYTES DECIMAL ";K$:IFK$="END"THEN4
954 P1=VAL(K$):IFA+P1>0ANDA+P1<=BYTETHEN966
956 PRINT"OUT OF EPROM ADDRESS RANGE      TRY AGAIN"
958 SOUND10,9:PRINTCHR$(27);"                          ":PRINTCHR$(27);
960 PRINTCHR$(27);"                               "
962 PRINTCHR$(27);:PRINTCHR$(27);"                                        ";
964 PRINT"    ";CHR$(27);:GOTO952
966 IFP1>4096THEN978
970 OUT202,0:X=PEEK(-17000):POKE-17000,99:Y=PEEK(-17000)
972 POKE-17000,X:IFY=99THEN992
974 X=PEEK(-15000):POKE-15000,99:Y=PEEK(-15000):POKE-15000,X
976 IFY=99THEN992
978 PRINT"RAM FAIL OR NO OF BYTES"
980 PRINT"SPECIFIED EXCEEDS RAM ADDRESS   RANGE.     TRY AGAIN"
982 SOUND10,9:PRINTCHR$(27);
984 PRINT"                          ":PRINTCHR$(27);
985 PRINTCHR$(27);
986 PRINT"                          ":PRINTCHR$(27);
987 PRINTCHR$(27);
988 PRINT"                          ":PRINTCHR$(27);
990 PRINTCHR$(27);:GOTO952
992 M2=A+P1-1:IFM2<16384THENP2=0:M2=M2+12*4096:GOTO995
994 M2=M2-16384+12*4096:P2=1
995 FORZ=1TO500:NEXT:M3=47104:PRINT
996 INPUT"PRESS RETURN WHEN READY";K$:IFK$="ENDTHEN4
997 IFM1=M2THEN4ELSEOUT202,37
998 F2=1
999 PRINT,,,"PROGRAMMING ";
1000 S0=(E1AND208)OR40
1002 IFBIT2=0ANDP2=1THENCOUNT=2ELSECOUNT=1
1004 M8=M1:M9=M2
1006 IFE0<>3THENS1=S0OR4:S2=S0:S3=S0OR6:GOTO1012
1008 IFCOUNT=1ANDP2=1THENA14=4ELSEA14=0
1010 S1=(S0ORA14)OR1:S2=S1AND254:S3=S1OR3
1012 POKE-20658,S1:POKE-20652,S2:POKE-20638,S1:POKE-20634,S3
1014 POKE-20621,S1
1016 IFP2=1ANDBIT2=0THEN1026
1018 H=INT(M1/256):L=M1-256*H:POKE-20674,L:POKE-20673,H
1020 H=INT(M2/256):L=M2-256*H:POKE-20671,L:POKE-20670,H
1022 H=INT(M3/256):L=M3-256*H:POKE-20667,L:POKE-20666,H
1024 GOTO1028
1026 IFCOUNT=2THENM2=65535ELSEM1=49152:M2=M9
1027 GOTO1018
1028 POKE-20678,1:POKE30862,60:POKE30863,175
1029 OUT202,S1:FORZ=1TO400:NEXT:X=USR(X):OUT202,37
1030 FORZ=1TO100:NEXT:IFPEEK(-20678)=1THEN1040
1032 SOUND10,5:PRINT"FAIL":SOUND10,5:OUT202,0
1034 PRINT"NO OF BYTES PROGRAMMED          BEFORE  FAILURE:";
1036 IFA14=0THENPRINTPEEK(-20680)+256*PEEK(-20679)-12*4096
1038 IFA14=4THENPRINTPEEK(-20680)+256*PEEK(-20679)-8*4096
1039 GOTO1150
1040 COUNT=COUNT-1:IFCOUNT=1THEN1008
```

4 of 5.

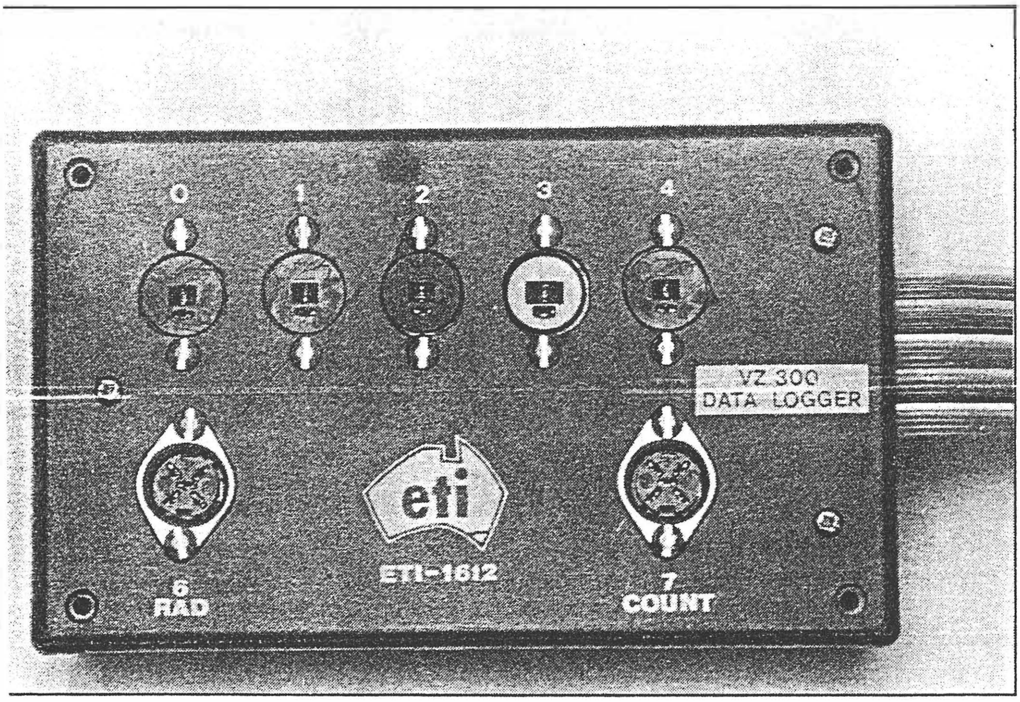
```
1042 M1=M8:M2=M9:IFBIT2=0ANDP2=1THENCOUNT=2ELSECOUNT=1
1044 S0=(S0AND128)OR32
1046 IFE0<>3THENS4=S0OR5:S5=S0OR6:GOTO1052
1048 IFCOUNT=1ANDP2=1THENA14=4ELSEA14=0
1050 S4=(S0ORA14)OR1:S5=(S0ORA14)OR2
1052 POKE-20585,S4:POKE-20574,S5:POKE-20561,S4
1054 IFP2=1ANDBIT2=0THEN1062
1056 H=INT(M1/256):L=M1-H*256:POKE-20595,L:POKE-20594,H
1058 H=INT(M2/256):L=M2-H*256:POKE-20592,L:POKE-20591,H
1060 H=INT(M3/256):L=M3-H*256:POKE-20588,L:POKE-20587,H
1061 GOTO1064
1062 IFCOUNT=2THENM2=65535ELSEM1=49152:M2=M9
1063 GOTO1056
1064 POKE-22678,1:POKE30862,140:POKE30863,175:X=USR(X)
1066 IFPEEK(-20678)=0THEN1032
1068 COUNT=COUNT-1:IFCOUNT=1THEN1048
1070 OUT202,0:SOUND15,5:PRINT"COMPLETE.":PRINT
1072 PRINT"VERIFY OK",,,,:SOUND10,5
1074 GOTO1150
1100 E0=E1AND3:IFE0=0GOSUB800:GOTO1100
1102 CLS:PRINT"COPY EPROM 2 TO EPROM 1"
1104 PRINT"--------------------------------"
1106 PRINT"DO NOT INSERT OR REMOVE EPROM   WHILE LED IS ON"
1108 PRINT:PRINT"EPROM TYPE",
1110 IFE0=1THENPRINT"     2764/27C64":GOTO1116
1112 IFE0=2THENPRINT"   27128/27C128":GOTO1116
1114 PRINT"   27256/27C256"
1116 PRINT"PROGRAMMING VOLTAGE VPP";:E=E1AND16
1118 IFE=16THENPRINT"      21V"ELSEPRINT"   12.5V"
1120 PRINT"VCC DURING PROGRAMMING";:E=E1AND64
1122 IFE=64THENPRINT"        6V"ELSEPRINT"        5V"
1124 PRINT:INPUT"IS THIS INFORMATION CORRECT,Y/N";K$
1126 IFK$<>"Y"ANDK$<>"YES"THENGOSUB800:GOTO1100
1128 IFE0<>1THEN1132
1130 BIT1=0:P2=0:M2=57343:COUNT=1:GOTO1138
1132 IFE0<>2THEN1136
1134 BIT1=0:P2=0:M2=65535:COUNT=1:GOTO1138
1136 BIT1=0:P2=1:M2=65535:0UNT=2
1138 S0=(E1AND80)OR168
1139 F2=2
1140 M1=49152:M3=49152:OUT202,37:FORZ=1TO500:NEXT
1142 PRINT"PROGRAMMING ";:GOTO1002
1150 INPUT"DO YOU WISH TO DO AGAIN Y/N";K$
1152 IFK$<>"Y"THENF2=0:GOTO40
1154 IFF2=1THENF2=0:GOTO900
1156 F2=0:GOTO1100
1300 A=PEEK(-18432+D):GOSUB400
1302 POKE-18432+D,99:IFPEEK(-18432+D)<>99THEN265
1304 POKE-18432+D,A
1310 PRINT,CHR$(27);RIGHT$("00"+A$,2);"
1315 GOTO285
```

5 of 5

ETI-1612 VZ300 Data Logger

For $60 you can build a box to plug into a VZ300 computer to log up to 8 analogue channels. Data can later be stored on cassette tape.

Bob Sutton



Specifications

Number of channels: 8 analogue (designated 0 through 7).

Channel 7 is used as a counter, being driven from an open collector transistor. Channels to be logged are selected by program.

Voltage Range: +2.5V (count 0) to +3.56V (count 255) with common O V.

Range can be hardware modified to any window in the range 0 to +5V.

Sampling Rate: 3 per second.

This is high enough to count up to 1 pulse per second on channel 7.

Calibration: Transducers are calibrated individually. Every 10 seconds a scan of channels appears on the screen.

Reliability: mainly determined by the reliability of the mains supply.

Power supply: +5V from the VZ300.

Averaging/Counting Interval: 1 hour. This can be changed by program.

Designated RAM Store: 6K bytes. This can be extended; each byte holds one value. 5 channels hours for 51 days fills 6K of RAM.

Digital outputs: There are three digital outputs which could be used for indicators, alarms or control.

THE TASMANIAN BRANCH of the ANZ Solar Energy Society needed a cheap means of recording temperatures and other variables in passively heated solar houses. About 10 days of hourly recording are required to be sure of getting the thermal thumbprint for a house. I thought of designing a battery-powered data logger around the Motorola MC146805 microprocessor but decided instead it would be faster to build an attachment for a cheap, mains-powered microcomputer and to program it in a high level language. Having recently taken a course on the Z-80 microprocessor with Scott Ashton at Elizabeth College I chose the Z-80 based VZ300 which sells for around $120. Of course a TV screen or monitor plus a cassette recorder are also needed. (This is not the first time a VZ has been used as a data logger: Bruce Baudinet of Sunspot Design built one for the VZ200.)

This article gives sufficient detail to build the box (called the "logger") to collect data, to store the data on cassette tape, to retrieve it and to plot a graph. As examples the logger and programs are for the configuration I use for solar work. The programs deliberately lack refinements so that someone literate in BASIC can modify them readily to suit other requirements. Examples of sensors/transducers and their interfacing are given.

I/O Operation

The VZ300 can transfer data from/to up to 256 input/output ports using the INP and OUT instructions. Data is transferred under the control of the $\overline{RD}$, $\overline{WR}$ and $\overline{IORQ}$ lines. I have designated the logger to be the vacant port 64. Thus the code Z=INP(64) transfers one byte (8 bits) of data from port 64 to the real variable Z. Likewise OUT 64,Y transfers Y to the logger output latch. Y can be a constant, a real variable, an integer variable or an

1 of 5.

Table 1: A/D control
Lower case letters are used to avoid confusion with the VZ300 lines

$\overline{wr}$	$\overline{rd}$	
1	1	dormant
1	0	offer converted
0	1	start conversion
0	0	forbidden

Table 2: VZ300 output port configuration showing start conversion and offer value instructions for channel 2.

spare	A/D	select	
7 6 5	4 3	2 1 0	LSB
	$\overline{wr}$ $\overline{rd}$	a1 a1 a0	
0 0 0	0 1	0 1 0	start conv= 8+2=19=OAH
0 0 0	1 0	0 1 0	offer value= 16+2=18= 12H

expression but it must be an integer in the range 0 to 255.

The latch (IC2) is used to select the analogue channel (lowest 3 bits) and to control the A/D converter (next 2 bits). The highest 3 bits are spare and their contents are irrelevant.

The five steps to collect a sample are:

1. SELECT the analogue input channel;
2. START the A/D conversion;
3. WAIT for completion;
4. OFFER the converted value to the VZ300;
5. INPUT to VZ300.

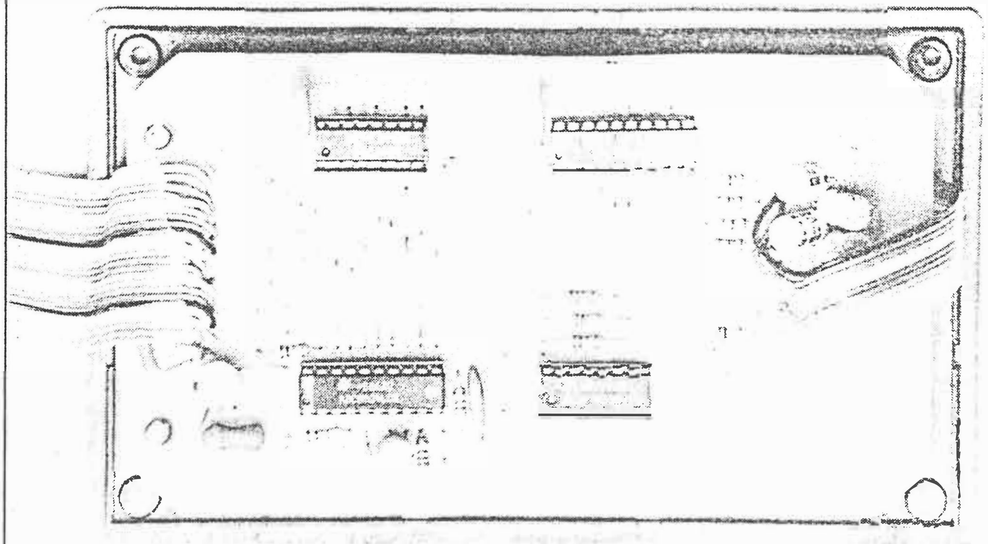
OFFER and SELECT can be combined when treating channels sequentially. Table 1 gives the A/D control and Table 2 gives an example of the START and OFFER patterns. Programs 1, 2 and 3 are suitable for testing.

Cassette Data Storage

The collection program (see box) POKEs data into a 6 K block of unused memory. This data is then stored on cassette tape by making the operating system think it is storing a program. Later the data is recovered by the reverse procedure and then some data processing program is loaded and run.

The following is the procedure to be followed to store and recover all 6 K. The modification for reduced storage is given later.

1. Load and run Program 4.
2. Terminate it at the end of logging by CTRL/BREAK.
3. Then type the following instructions, terminating each with RETURN.
   POKE 30884,254
   POKE 30885,143
   POKE 30969,0
   POKE 30970,168
4. CSAVE"datname" having started the tape recorder before RETURN.
5. Choose your own "datname".



Converting to VZ200 operation

With only program modifications the logger will work with the earlier VZ200. The VZ200 has a 3.58 MHz clock, compared with the VZ300 at 3.54 MHz. Therefore some adjustments may be desired in lines 430 and 470 of Program 4.

The main difference lies in the available storage. The VZ200 has a 6K RAM whereas the VZ300 has 16K. With the following changes the VZ200 will run a program as large as Program 4 in conjunction with a 2K data store:

Program 4: in line 330 put −31232
in line 840 put −29184
Immediate POKES: POKE 30884,254
POKE 30885,133
POKE 30969,0
POKE 30970,142
Program 5: in line 30 put 2048 twice
in line 40 put −31232
Program 6: in line 70 put −31232

Continue reading this section only if you want to run large processing programs or if you require more than 2K of data store. Refer to the memory maps starting at the RAM. In both computers the program extends above location 31465, first with the BASIC code and then the numeric variables. String variables and the "stacks" extend downwards from the top of store. The spaces between are free for data storage. I started the VZ200 store at location 34304 = 8600H. For POKE and PEEK instructions the locations above 32767 (= 32K −1) are addressed using negative integers (64K being zero). For example 34304 = -31232. You can search for free space by typing NEW and then using something like Program 5.

As checks of the extents of program and variables it is useful to examine the contents of the address pointers. These two-byte pairs contain the relevant addresses, always starting with the low order byte. For example the BASIC program starts at location 31465 = 7AE9H. Thus from the list of pointers 30884 contains 233=E9H and 30885 contains 122=7AH; this may be verified using PEEKs. At startup, before any program has been entered, the end-of-basic is just two bytes further on at 31467. As program is loaded the end-of-basic advances.

Pointers	Hex	Decimal
End of stack (= start of strings)	78A0/1	30880/1
Start of dimensioned variables	78FB/C	30971/2
End of BASIC	78F9/A	30969/70
Start of BASIC	78A4/5	30884/5

The VZ300 is supplied with a 12V battery eliminator instead of a 9V one. The extra voltage drop tends to overheat the VZ300 voltage regulator. With the extra current drawn by the logger this situation is made worse. A high wattage series resistor may fix this. Instead I used a slightly under-rated 9V battery eliminator and initially got random variations in A/D conversions due to 100 pps negative bumps on the 5V rail. A capacitor across the 9V leads cured this.

2 of 5.

6. Switch the computer off and then on again before reloading data.
7. To reload data and process
   switch on
   CLOAD"datname"
   NEW
   CLOAD"processprog"
   RUN
   To store less than 6 K, change the 168 in POKE 30970,168 above, to 144 + the number of blocks of 256 bytes (including partly filled blocks). For example if 5 channels were logged hourly for 190 hours then there would be 950 bytes and therefore 4 blocks would be required. Thus the number would be 148 instead of 168.

Analogue Circuits

The ADC0804 A/D converter features span adjustment and high impedance differential input. The inputs have diode clamps which with high source resistance hold the input voltages in the required range of −0.3 V to +5.3 V.

The span control Vref/2 at pin 9 appears from the outside as a 2.5 V source in series with about 1000 ohms. External resistors are added to alter the pin 9 voltage. The span is twice the voltage at pin 9.

The converted count is given by
$C = (V^- — V^-) \times 128 / Vpin\ 9$
For example when $V^+ = +3.1$, $V^- = +2.5$ and Vpin9 = Vref/2 = 0.5, the count is 153. Out-of-range inputs give counts of 0 or 255.

Transducers

For temperature measurement I mostly use the LM335 sensor. Provided it passes at least 0.5 mA it behaves as a temperature controlled zener diode. The constant is nominally 10 mV/K. Thus at 0°C (=273.2 K) the nominal voltage is 2.73 V and at 30°C it is 3.03 V. The board has

Program 1: I/O Selector Test

To pulse low pin 11 of 74LS138

```
10 Y=INP(64)
20 GO TO 10
```

Program 2 Output Latch Test

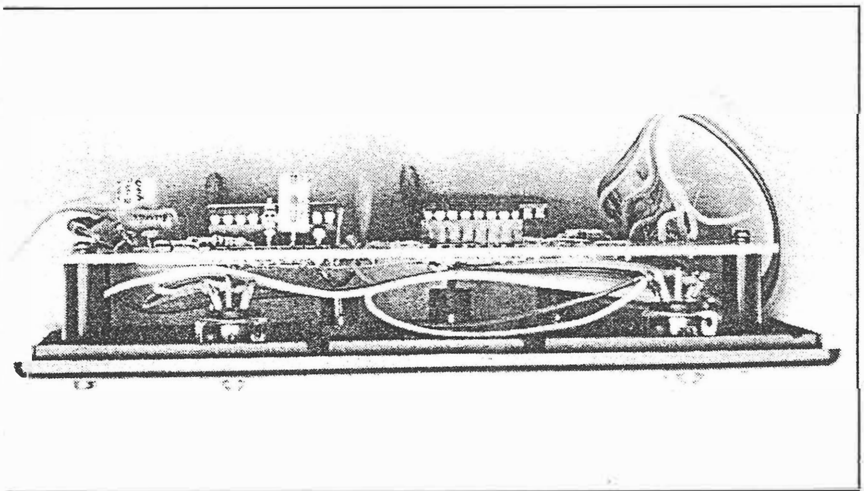
To continually output the number A% to the latch. The lowest 3 bits select the analogue inputs. Pin 13 of 74LS138 pulses low.

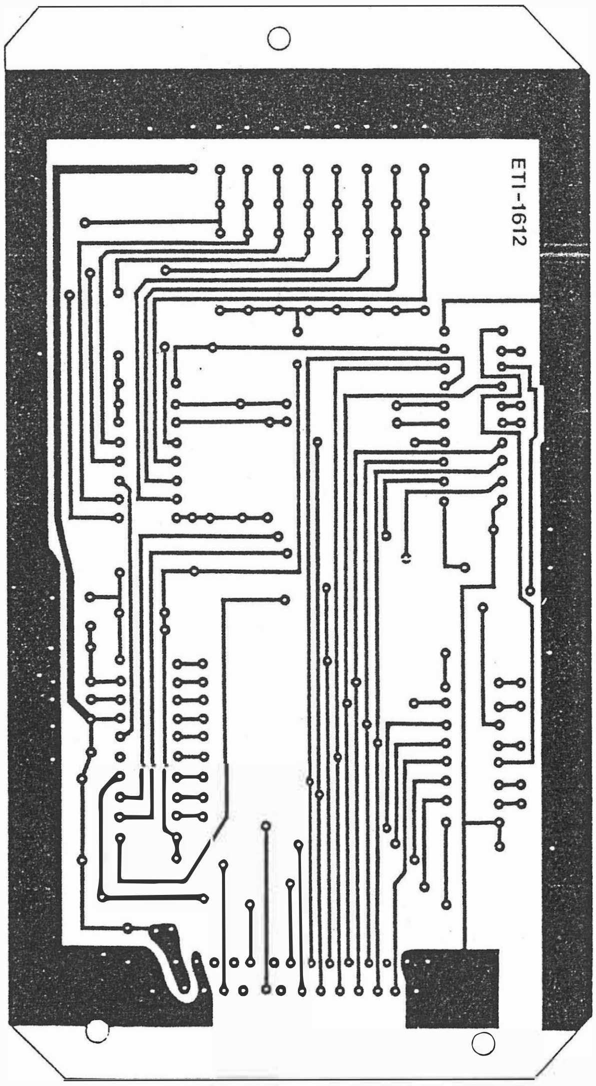
```
10 INPUT"INTEGER IN RANGE 0 TO 255";A%
20 OUT 64,A%
30 REM OPTIONAL DELAY
40 FOR I=ITO 200:NEXTI
50 GO TO 20
```

PROGRAM 3 SINGLE CHANNEL DISPLAY

To display a channel (0 to 7)

```
10 INPUT"CHN NUM";A%
20 OUT 64,24+A%     select channel
30 OUT 64,8+A%      start conversion
40 D=INP(64)        delay
50 OUT 64,16+A%     offer convtd value
60 PRINT INP(64)    input & print
70 GO TO 30
```






```
PROGRAM 5    VIEW DATA
This processing program just displays on the screen the raw values
retrieved from cassette tape.
10 INPUT"NUM OF PERIODS";N
20 INPUT"NUM OF ACTIVE CHNS";M
30 IF N*M)6144 THEN N=INT(6144/M)
40 AP=-28672
50 FOR I=1TO N
60 PRINT I;
70 FOR J=1TO M
80 PRINT USING" ###";PEEK(AP)
90 AP=AP+1
100 NEXT J
110 PRINT
120 NEXT I
PROGRAM 6   PLOT DATA
10 CLS:MODE(1):COLOR 4
20 FOR Y=0TO50:SET(10,57-Y):NEXTY
30 FOR Y=0TO50STEP5:SET(11,57-Y):NEXTY
40FORY=0TO50:SET(107,57-Y):NEXTY
50FORY=0TO50STEP5:SET(106,57-Y):NEXTY
60FORX=10TO107:SET(X,57):NEXTX
70AP=-28672
100FORI=101TO150
110Y0=PEEK(AP+2*I-2)
120Y0=INT(.34*Y0+10.2+.5)
130COLOR 4:SET(10+I,57-Y0)
140Y1=PEEK(AP+2*I-1)
150Y1=INT(.31*Y1+8.4+.5)
160COLOR 2:SET(10+I,57-Y1)
190NEXTI
200GOTO200
This is broken by CTRL/BREAK
```

Program listings
All the program listed in this article are available on tape from: Tasmanian Branch ANZ5S5, PO Box 121, Sandy Bay, Tas 7005. Send $10 plus stamped self-addressed envelope.

3 of 5.

provision for pullup(/down) resistors and filter capacitors.

My photovoltaic solar radiation transducer gives about 300 mV full output which is quite compatible with the span for the LM335. The negative wire is simply joined to V^- and kept well insulated.

I measure electricity consumption by detecting the mark on the rotating disc of a kWh meter. This is done using a reflective opto switch (RS stock No. 307-913) costing about $15. The instrument has LEDs to indicate status to assist in aligning it on the glass in front of the disc. Rubber bands and self adhesive picture hooks are convenient for attachment. A 0.5 second pulse lengthener is required to ensure that a pulse is not missed when the disc is rotating quickly. The program counts pulses by detecting low-to-high transitions for channel 7. Because the IR LED alone draws 40 mA this instrument should be connected to other than the VZ300 +5 V supply.

Graphs

The VZ300 has two graphics modes: MODE (0) for text — 32 characters wide by 16 down (the default mode) and MODE (1) which is 128 x 64. The rectangle is the only symbol in MODE (1) but variation can be obtained by altering the shading.

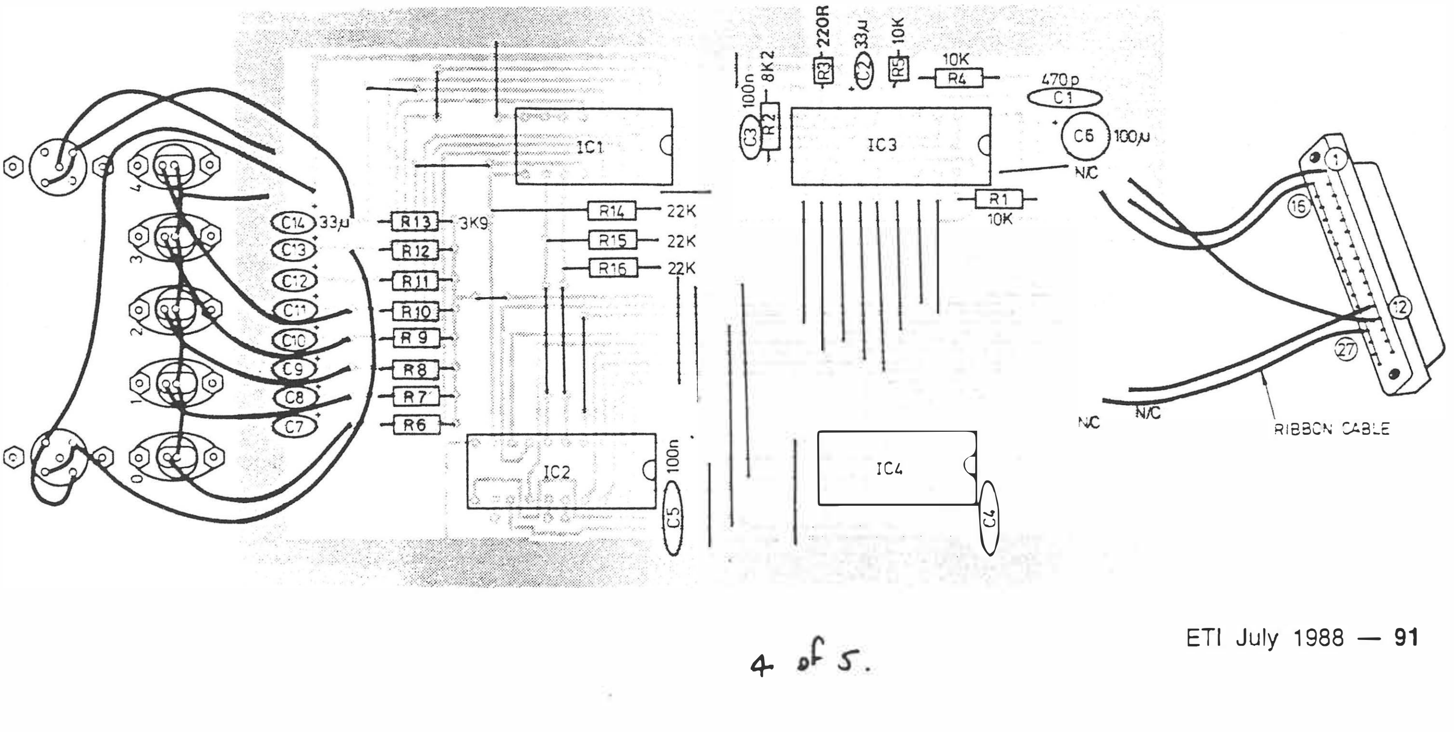
The SET(X,Y) instruction in MODE

Program 4 COLLECTION PROGRAM

```
            DATA COLLECTION

10 PRINT"DATA COLLECTION PROGRAM"
20 PRINT
30 DIM A(7),B(7),C$(7),L%(7),S(7)
100 REM INITIATE CONSTANTS, TIME, DATE
110 PRINT"CHANNELS"
120 PRINT"   SLOPE OFFSET IDENT"
130 FOR I=0TO7
140 READ A(I),B(I),C$(I)
150 PRINT USING"  ###.##";A(I);B(I);
151 PRINT C$(I)
160 NEXT I
170 PRINT"IF WRONG THEN BREAK & CHANGE";
171 PRINT" LINES 200-270"
180 PRINT"WRITE DOWN CORRECTED VALUES"
200 DATA 1,0,TEMP
210 DATA 1,0,TEMP
220 DATA 0,0,V
230 DATA 0,0,V
240 DATA 0,0,V
250 DATA 0,0,V
260 DATA 1,0,RAD
270 DATA 1,0,KWH
280 INPUT"NEXT HOUR OF DAY";H
290 INPUT"DAY OF MONTH";D%
300 PRINT"PRESS S TO START LOGGING"
310 A$=INKEY$
320 IF A$<>"S" THEN GO TO 310
330 SH=H:SD%=D%:AP=-28672
335 POKEAP-2,255:POKEAP-1,254

340 IF H<23.5 THEN GO TO 400
350 H=0:D%=D%+1
400 FOR K=1TO360
410 FOR L=1TO30
420 GOSUB600:REM SCAN
430 FOR D=1TO5:NEXT D:REM DELAY
440 NEXT L
450 REM PRINT HOUR & ACTIVE INPUTS
451 GOSUB700
470 FOR D=1TO39:NEXT D:REM FINE DELAY
480 NEXT K
490 REM TRANSFER ACTIVE CHN AVERAGES TO RAM
491 GOSUB800
500 H=H+1
510 GO TO 340

600 REM SUB SCAN
605 OUT64,24
610 FOR I=0TO7
615 OUT64,8+I
620 D=INP(64)
625 OUT64,16+I
630 L%(I)=INP(64)
635 NEXT I
640 FOR I=0TO6
645 S(I)=S(I)+L%(I)
650 NEXT I
655 IF L%(I)>128 THEN NW=1 ELSE NW=0
660 IF NW>OL THEN S(7)=S(7)+1
665 OL=NW:L%(7)=INT(S(7))
670 RETURN

700 REM SUB PRINT LATEST
710 PRINT D%;H;
720 FOR I=0TO7
730 IF C$(I)="V" THEN GO TO 750
740 PRINT L%(I)*A(I)+B(I);
750 NEXT I
760 PRINT
770 RETURN

800 REM SUB STORE
805 FOR I=0TO7
810 IF C$(I)="V" THEN GO TO 860
815 XD=S(I)/10800
820 IF I=7 THEN XD=XD*500
825 X%=INT(XD+.5)
830 IF X%>255 THEN X%=255
835 S(I)=0
840 IF AP>=-20480 THEN STOP
845 POKE AP,X%
850 PRINT X%
855 AP=AP+1
860 NEXT I
865 RETURN
```

The collection program has the following features:

1. **All 8 channels are sampled three times a second. Values from channels 0 through 6 are accumulated to be divided by 10,800 after an hour to give average values. Channel 7 (counter) is accumulated and effectively divided by 21.6 so that it can never overload.**
2. **Each hour, values for active channels are transferred sequentially to storage in RAM starting at address 36864 = 9000H. An active channel is one without a "V" (for vacant) in lines 200 to 270.**
3. **At initialisation the user enters the starting hour (integer 0 through 23) and the day of month. Sampling commences when "S" is pressed. The user determines the significance of the hour eg, period starting, or centered on, or finishing.**
4. **Logging is terminated by CTRL/BREAK or when the store fills. Data for the unfinished hour is lost.**
5. **Day of month is sequential but does not revert to 1 at any change of month.**
6. **Every 10 seconds the screen receives the latest day, hour and scaled values for active channels. This is useful for monitoring and calibrating. Scaling is multiplying by the appropriate constant and adding the offset stored in lines 200 to 270.**



4 of 5.

(1) marks the rectangle at the position X (across), Y (down). To get normal plots with Y positive up the variable effectively becomes 63-Y.

Program 6 draws axes and then plots scaled values of data for two channels for time intervals 101 to 150. Lines 120 and 150 contain the appropriate scaling formulae; the +.5 being for correct rounding. A natural improvement would be to store the scaling constants and list of active channels in arrays as in Program 4; but the aim here is to keep it simple.

Construction

Construction is straightforward and only a logic probe is needed for any trouble shooting.

Decide on your input socket layout and then mount suitable polarised sockets on the lid of the box (We used two pin DIN sockets in the prototype.) To minimise crosstalk, keep the common side resistance low in the cable to the board. Also leave the cable long enough to allow the sections to be separated for testing. Solder the passive components — links, capacitors, resistors and IC sockets. Install plenty of test pins. Finally add the 25 way ribbon and 30 way socket to the VZ300 printer port. Solder the only crossover first (socket pin 12); then solder all other pins sequentially (1, 16, 2, 17, . . .). File a depression in the box to hold the ribbon firmly with the box shut. Visually and using an ohm meter check for shorts between adjacent tracks.

Testing

ALWAYS SWITCH OFF THE COMPUTER BEFORE PLUGGING/UNPLUGGING THE LOGGER OR ADDING/REMOVING IC'S.

First, with no logger IC's test that the computer keeps working and that the +5 V reaches all sockets. A logic probe would indicate activity on the address and data lines.

Refer to the section on I/O operation.

Second, insert the I/O selector (74LS138), run Programs 1 and 2 and check separately for low pulses on pins 11 and 13. You will need a logic probe to pick up the pulses. If a logic probe is unavailable then proceed anyway.

Third, insert the data latch (74LS374) and check that it correctly accepts bit patterns from the computer. A voltmeter can be used.

Fourth, taking the usual precautions to earth yourself and the board, inset the analogue selector (CD4051) and test for the output signal at pin 3. Select channels by program via the latch. The analogue inputs have pullup resistors so operation can be checked by earthing inputs.

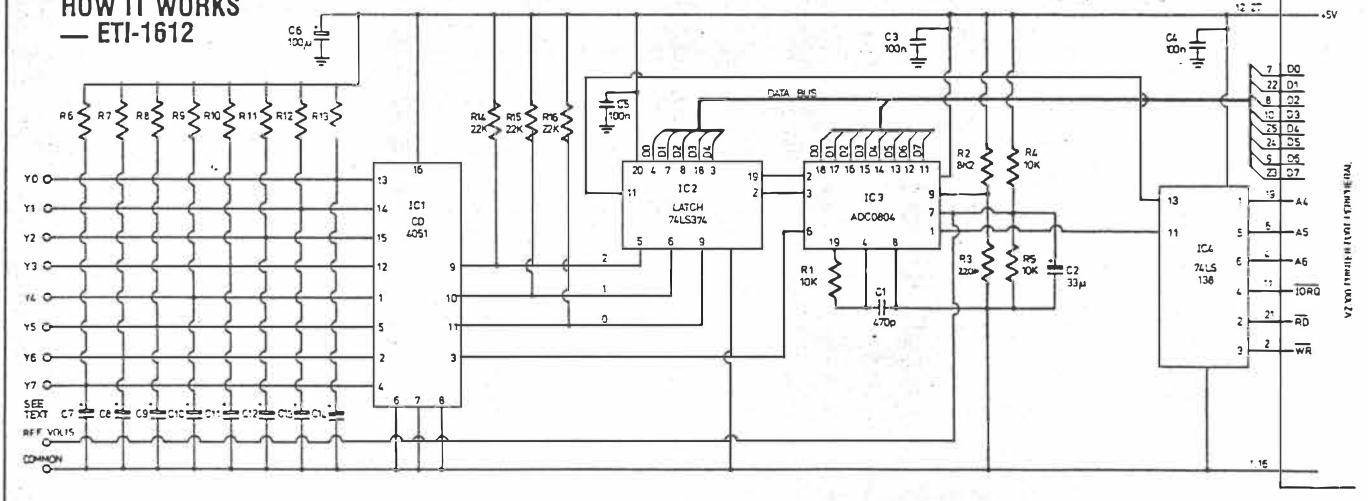
PARTS LIST — ETI-1612

Resistors	All ¼W unless otherwise stated.
R5, R1, R4	10k
R2	8k2
R3	220R
R6-R13	see text
R14, R15, R16	22k
Capacitors	
C1	470p cer
C2	33µ/10V
C3, C4, C5	100n cer
C6, C13	see text
C14	33µ electro
Semiconductors	
IC1	4051
IC2	74LS374
IC3	ADC0804
IC4	74LS138N
IC5	LM335

Miscellaneous
5 x 2 pin Din, 3 x 5 pin Din sockets, ribbon cable, hook-up wire & box.

Fifth, again taking care with earthing, insert the analogue-to-digital converter (ADC0804LCN). Check for oscillator action — pin 4. The analogue voltage reference (pin 7) should be around 2.5 V and the span voltage (pin9) around 0.53 V. Run Program 3 to test the logger. Then proceed to full data collection — Program 4. To display scans more frequently than evey 10 seconds reduce the 30 in line 410.●

HOW IT WORKS — ETI-1612



The logger is controlled from the VZ300 output port. Address lines A4 to A6 select the latch IC4, and the read and write lines drives either pin 11 or pin 13 active. These two outputs are connected to either the latch, IC2 or the converter IC3.

Data comes into the input port from one of seven channels in analogue form. The exact form of the transducer responsible for this is up to you. The input port is connected directly to a 4051 which functions as an analogue switch, so that it will take the analogue input and place it on the output pin, pin 3. Notice that space is provided for pull up resistors and capacitors on the input lines (Y0-Y6) which should be matched to the transducer. With an LM 335 temperature sensor, a 3.9 k resistor and 33µ capacitor are appropriate.

Which channel is selected depends on the configuration of pins 9, 10 and 11 on IC1. These are derived from IC2, which loads from the VZ 300 data bus when pin 11 is activated by IC4.

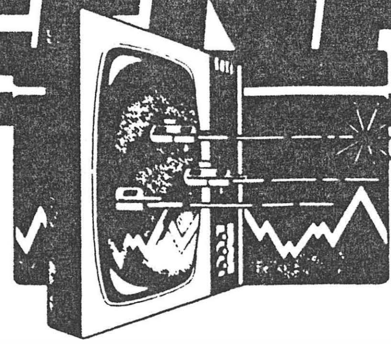
the ADC (pin 6, IC3). The ADC is controlled by pins 1, 2 and 3 and eventually the 8 bit converted value is transferred to the VZ 300 data bus, where it is read by the computer. R4, R5 and C2 set up the reference voltage for the ADC, and R2, R3 set the span. R1 and C1 trim the internal oscillator. Note that the reference voltage is available to the external world via the channel seven socket.

5 of 5.

COMMERCIAL SOFTWARE REVIEWS

Mar.	84	APC	190-1	Review of DSE 'Matchbox', 'Biorhythms', 'Circus' and 'Poker'. (Davies)	(2)
Aug.	84	PCG	46-47	Review of DSE 'Panik' and 'Ladder Challenge'.	(1)
Oct.	84	PCG	90-91	Review of DSE 'Knights and Dragons', 'Ghost Hunter', 'Othello', and 'Invaders'.	(2)
Nov.	84	PCG	90-96	Review of LYSCO 'Cub Scout' and DSE 'Dracula's Castle'.	(1)
Jan.	85	PCG	65	Review of DSE 'Air Traffic Controller' and 'Tennis'.	(1)
Feb.	85	PCG	76	Review of DSE 'Defence Penetrator' and 'Star Blaster'.	(1)
Mar.	85	PCG	76-77	Review of DSE 'Planet Patrol' and 'Learjet'.	(1)
Apr.	85	PCG	94-99	Review of DSE 'Asteroids', Super Snake' and 'Lunar Lander'.	(1)
Apr.	85	ETI	103	Logbook and Morse on VZ-200.	(1)
Oct.	85	PCG	68-9	Review of DSE 'Duel'.	(1)
Nov.	85	PCG	70-1	Review of DSE 'Attack of the Killer Tomatoes'.	(1)
Nov.	85	CLC	31	Review of educational software.	(1)

SCREENPLAY



Ian Davies has a look at games for Dick Smith's VZ-200.

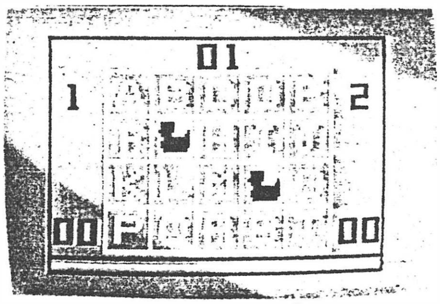
MATCH BOX

Game: Match Box
Supplier: Video Technology
Price: $12.50

Match box is a memory enhancement program designed to increase your power of recollection in a game format. It runs on a standard VZ-200 with no extra memory required.

The screen is divided into twenty-five squares, each identified by a single letter. Beneath each square is a hidden symbol. Two players are required for this game, and the computer will take it in turns asking each player to select a pair of squares. The symbols underneath these squares will be revealed briefly, and then hidden again.

The objective of the game is to match up as many identical pairs of symbols as possible, and so it is necessary to remember where various symbols have appeared. Once a pair of symbols have been involved in an identical match, they are thereafter out of play. Each match scores a player one point, and the player with the highest number of points wins the game.



Match box is a series of three basic programs, which are automatically loaded into the VZ-200 one after the other. The first program displays the name of the game, the second provides instructions and the third actually plays the game. Because of this, Match Box is painfully slow to load and cannot repeat the instructions after a game without completely re-loading all three programs.

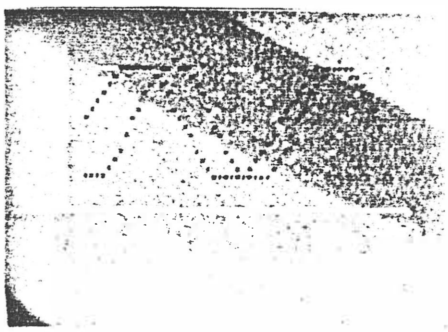
Additionally, the game runs very slowly and seems to crash regularly – requiring a complete re-load. On the plus side, Match Box will help to increase your retention and is non-violent – two rare characteristics in video games.

Use of graphics:	★★★
Use of sound:	★★
Addictive quality:	★★
Game speed:	★
Use of colour:	★★
Value for money:	★★

BIORYTHMS

Game: Biorythm/Pair Matching/ Calendar
Supplier: Video Technology
Price: $12.50

This tape consists of three programs all related to dates. The biorythm program (pictured) predicts your emotional, physical and intellectual highs and lows over a given period. It does this in a graphical format and provides text to (incorrectly) explain the meaning of the graph.



The pair matching program accepts the birthdates of *two* people and then tells you which week day they were born on. It then goes on to produce a percentage of compatability for emotional, physical and intellectual factors. It does this by comparing the two biorythms involved – a trivial process based on the number of days between the two dates.

The calendar program accepts two dates and tells you which day of the week those two dates were, and also how many days are between the two dates.

If you are thinking that these programs apply the same simple formula in three different ways, then you are probably correct. They all perform useful functions, but do not perform anything particularly clever.

Use of graphics:	
Use of sound:	
Addictive quality:	★
Game speed:	★
Use of colour:	
Value for money:	★

APC Mar 84 5(3) p 190-191
1 of 2.

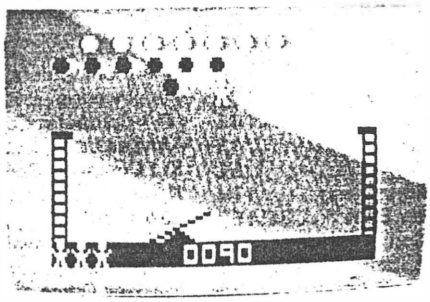
CIRCUS

Game: Circus
Supplier: Video Technology
Price: $12.50

Under the Circus Big Top, acrobats perform death defying stunts on the catapult (see-saw). One acrobat jumps off a high platform onto the empty end of the catapult, thereby sending the other acrobat flying high into the air.

Your job is to move the catapult from left to right so that the acrobats continue to land on their respective ends and project the other into the air. A stream of balloons float high above the ring, and the acrobats must collect as many of these as possible for ten points per balloon. The game becomes progressively faster until it runs at an impressive speed, thereby sorting out the men from the boys.

Control of the catapult is really rather complex, as the game accurately models



the actions of a real catapult. In other words, the second acrobat will be projected differently depending on how close to the pivot point the first one lands. This type of subtle control is very important, as the player inevitably finds himself in a position where the falling acrobat is going to land on top of the other acrobat. The only alternative is to move the catapult completely out of the way, in which case the airborne acrobat falls to his doom. With careful control, the dedicated player can learn to avoid this situation.

The game is over either when all the balloons have been collected, or when there have been five fatal falls.

Circus runs on an unexpanded VZ-200 and is played to the tune of "My Body Lies Over The Ocean". The game can make use of a joystick if one is installed. In general, Circus is a great deal of fun and rather addictive until one has master control of the catapult.

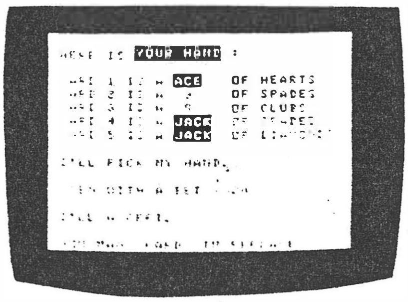
Use of graphics: ★★★★
Use of sound: ★★★
Addictive quality: ★★★★
Game speed: ★★★★★
Use of colour: ★★★
Value for money: ★★★★

POKER

Game: Poker
Supplier: Video Technology
Price: $12.50

VZ-200 Poker is a rather sad implementation of straight draw poker – you against the computer. It allows you to bet, raise, call, bluff and fold. So much for the good news.

Poker is written in Basic, and makes absolutely no use of colour, graphics or sound. These sins could easily be forgiven if it was a particularly good poker player, but alas, it is not. The program suffers badly from a fear of large bets, so a 100% reliable way to win is to place a bet of $100. It will



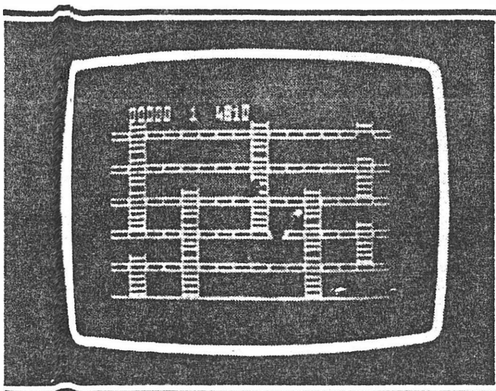
immediately fold.

The program will happily replace –3 of your cards, and will even replace the same one three times. Any non-numeric input will result in the familiar "?REDO" message from Basic. The player can happily continue to spend more money than he owns.

Poker is the type of game that any novice could write in a single evening after a few weeks experience with Basic. That a game of this quality is available for purchase is disappointing.

Use of graphics:
Use of sound:
Addictive quality: ★
Game speed: ★
Use of colour:
Value for money: ★

APC Mar 84 5(3) : 190-191
2 of 2.



GAME: Panik

MACHINE: VZ-200

JOYSTICK: Optional

SUPPLIER: Dick Smith

PRICE: $12.50

OVERALL: ***

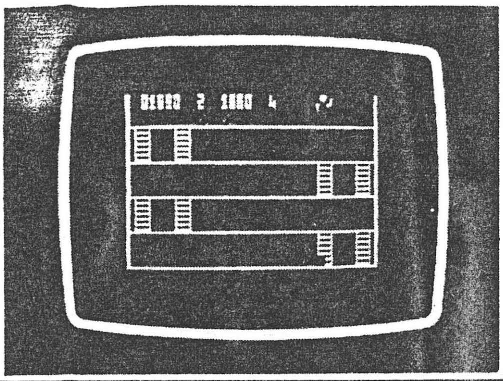
The object of the game is to free yourself from a six storey building which is besieged with hungry man eating monsters. The only way to stop the monsters is to dig holes in the floor and wait for a monster to fall into one of them. The monsters are then killed by hitting them over the head with your shovel.

You have to watch out for other monsters coming up from behind to attack you, while you're busy hitting his friend with your shovel.

As the game proceeds, you must make the monsters fall two or three floors to kill them. The number of monsters also increases per frame.

A highly recommended game.

PCG. Aug 84 p 43.



GAME: Ladder Challenge

MACHINE: VZ-200

JOYSTICK: Optional

SUPPLIER: Dick Smith

PRICE: $12.50

This fast moving game shows some resemblance to the arcade favourite Donkey Kong. The first frame of this four frame game is a series of ladders that you must negotiate, and fast moving barrels that you must jump over as they roll towards you.

In the second frame, a series of boxes move along various conveyor belts, the object being to reach the top ladder by dodging or jumping over the moving boxes. As these boxes move at only half the speed of the barrels, they must be negotiated "on the run" by the simultaneous use of the left and right controls and the jump button. There is a risk, however, that you may hit your head on an overhead box and be killed. It is possible to reach the top ladder without jumping by simply running up and down the ladders to dodge the boxes, however this technique can be slow resulting in a low point score.

In the third frame you must reach the top via four elevators, avoiding robots that guard each floor. The robots can be fended off with "energy shields", activated by the fire (jump) button, however these must be used sparingly as you only have a limited number available.

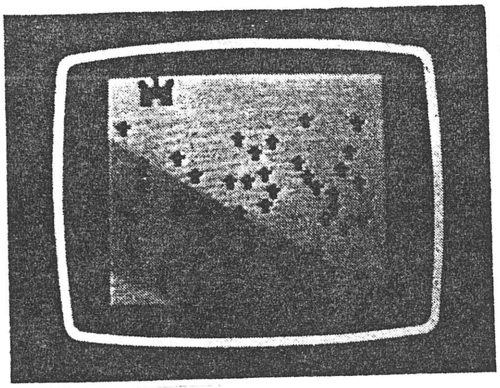
If you graduate to the fourth frame you are presented with a series of red pins that you must remove by simply walking over them, however your movement is once again severely restricted by robots that protect each floor:

Points are awarded at the end of each frame and are based on the time taken to reach the top. The time remaining from a countdown starting at 5000 is added to your score at the end of each frame. Bonus men and energy shields are awarded during the game.

This fast moving game makes excellent use of the VZ-200 graphics, and becomes quite addictive. The only criticism of the game is in the instructions, five spelling errors being found within the six screen pages of text.

GRAPHICS	****
SOUND	***
ORIGINALITY	***
LASTING INTEREST	****
OVERALL	****

PCG. Aug 84 p 46-47



GAME: Knights and Dragons
MACHINE: VZ-200
JOYSTICK: No
SUPPLIER: Dick Smith
PRICE: $12.50

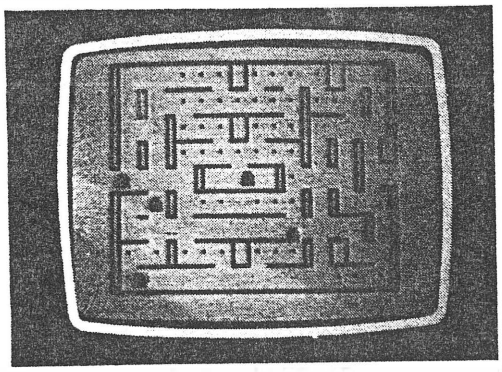
Knights and Dragons takes place in Medieval England. You are a Knight and are placed in a dragon's forest, unarmed. By using skill and cunning alone you need to return to your castle. If you succeed the castle baron will reward you with a purse of 100 gold coins. However, if you fail . . . the death march is aptly played.

The screen shows a castle in the top left corner and the rest of the screen is filled with randomly placed trees. You are placed in the forest and use the four arrow keys to avoid the dragon and to get to the castle.

The dragon is very cunning and often looks as if it's stuck in a tree. However, sure enough as you move closer to the castle he makes his timely charge and more often than not, he wins. When the dragon has eaten you the death march is played and on the screen is displayed "You have just become a square meal. Do you wish to be killed again?" Swallow your pride, type in 'Yes' and away you go.

There are five levels of difficulty to choose from. This varies the amount of trees on each screen. The graphics could be improved and the sound is limited. Nevertheless, the game was fun to play but would become easy after a while. Children will love it. **MB**

GRAPHICS	**
SOUND	**
ORIGINALITY	***
LASTING INTEREST	***
OVERALL	***



GAME: Ghost Hunter
MACHINE: VZ-200
JOYSTICK: Optional
SUPPLIER: Dick Smith
PRICE: $12.50

Hate to say it, but here's another *Pac-Man* clone. What more can be said about it that hasn't been said before?

For the new recruits to the maze-age the game is quite simple, but very clever.

In *Ghost Hunter* you have to avoid the ghosts and eat all the dots in the maze.

There are four powder pills, one in each corner — they are the large flashing dots. Eating one of these will allow you to chase the ghosts. When the screen turns to its original colour your chase time is up. After the third frame a bonus symbol will randomly appear below the ghost cage. If it is not eaten in time then the walls of the maze will disappear.

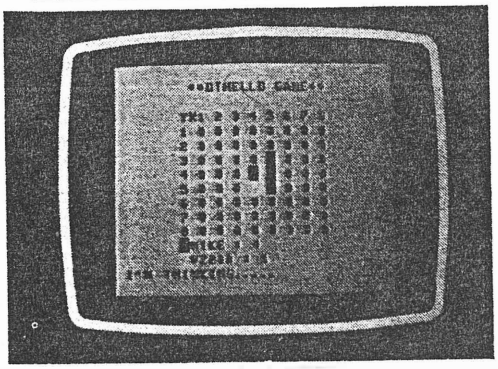
Not much can be said about the graphics as they don't change very much. Maze games are rather limited in their graphics. The colours are at least bright and are well contrasted against a lurid green background.

Maze games take a while to get used to if you haven't played them before. It's not so much the game but co-ordinating your fingers on the keys (that is of course if you're not fortunate enough to own a joystick). Yes it's frustrating, but isn't that why we're here — to overcome this and to beat the nasties?

Overall, a great game if you're a *Pac-Man* freak, if not, leave it. **MB**

GRAHICS	**
SOUND	**
ORIGINALITY	*
LASTING INTEREST	**
OVERALL	**

PCG. Oct 84 : 90-91
1 of 2.



GAME: Othello

MACHINE: VZ-200

JOYSTICK: No

SUPPLIER: Dick Smith

PRICE: $12.50

Othello is played on an 8 x 8 playing board similar to chess or checkers. The game starts off with each player having two pieces placed in the centre of the board. Each player in turn places one of his pieces on the board, in doing so capturing some of his opponent's pieces. At the end of the game, the person with the most pieces wins the game.

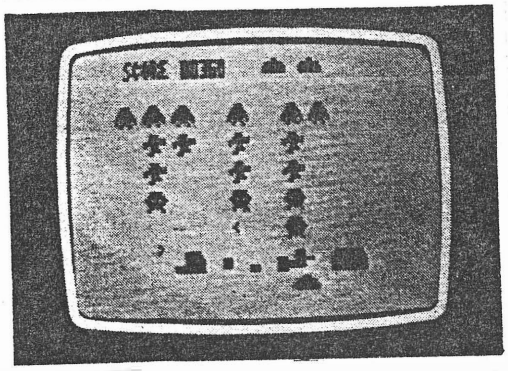
As always, this type of game requires forethought and strategic planning before executing your move. Pieces are only captured in a straight line but it will be either vertical, horizontal or diagonal. In many moves, pieces are captured in several different directions at once. You must however, capture at least one enemy piece per move. If there is no move that allows you to capture a piece, you must pass.

You can play against the computer and see how you do, which is what I did. I must confess, however, it's a daunting experience. Either I'm lacking intelligence or the computer cheats. I tried my best to execute a move which I believed to be fair and acceptable, but the computer just wouldn't accept it.

Othello is a great game for a rainy afternoon. Pack away the Monopoly and make way for a little logical thinking to bend and stretch those cerebral muscles.

Overall I found the game challenging, as any game of this type is. Frustrating it is, but well worth the effort. **MB**

GRAPHICS	*N/A*
SOUND	N/A
ORIGINALITY	*
LASTING INTEREST	***
OVERALL	***



GAME: Invaders

MACHINE: VZ-200

JOYSTICK: Optional

SUPPLIER: Dick Smith

PRICE: $12.50

Invaders from Dick Smith is based on the old game *Space Invaders*. Like the original it has nine frames to complete a game cycle but unlike the original the second and third cycle are made harder by increasing the number and speed of the missiles fired at you by the invaders.

For those of you who are new to *Space Invaders*, the game begins with a horde of aliens (or Invaders) who stomp across the screen back and forth. You have four defence barriers which slowly wear away after alien missiles (and your own) hit them.

The aliens slowly begin to move down the screen and the more you kill the faster they move.

There are three types of aliens: two rows worth 10 points each, two rows worth 20 points each, and one row worth 30 points each.

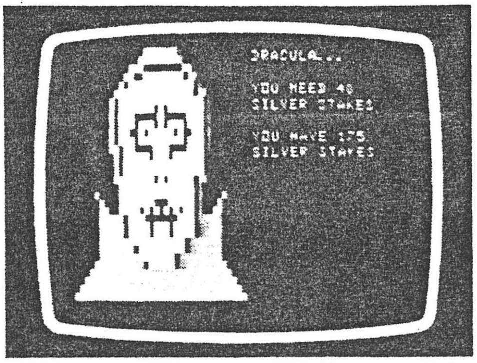
You only have three lives so you must try and kill all the aliens before you use up all three of your lives.

The game can be played with a joystick or keys may be used. The graphics naturally aren't as good as the original arcade game, yet are adequate.

The sound is limited to when you lose a life; the screen flashes and a high pitched noise emanates from the computer.

The game gets quite addictive but repetitive. It would be more suitable for the younger ones. **MB**

GRAPHICS	**
SOUND	**
ORIGINALITY	*
LASTING INTEREST	***
OVERALL	**



GAME: Dracula's Castle
MACHINE: VZ-200 (Expanded)
JOYSTICK: No
SUPPLIER: Dick Smith
PRICE: $12.50
OVERALL: ***

The aim of this adventure game is to get safely out of Dracula's Castle with as many silver stakes as possible, plus Dracula's treasure.

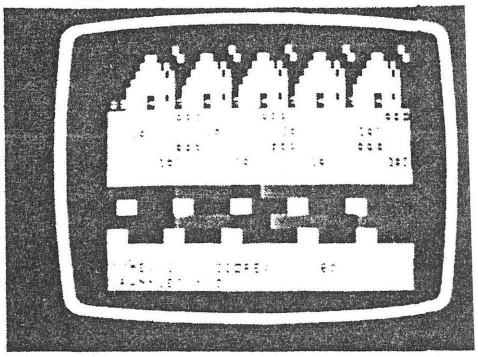
You move around the maze collecting silver stakes while avoiding Ghouls, Zombies, Slime Pits, Plant Creatures, Grells and of course, Dracula.

When you enter his castle, it is 30 minutes to sunset. Dracula rises at sunset and comes after you. You must use your silver stakes to kill the monsters.

As the game can become quite long to play, you are offered the chance to save the game in progress on tape so that it can be continued at a later date.

Overall, the game is quite entertaining and is value for money. IT

PCG Nov 84 1(4)
p. 90, 91, 96.



GAME: Cub Scout
GAME: VZ-200 (8k unexpanded)
JOYSTICK: Optional
SUPPLIER: Leon Young Software
PRICE: $8.00

Cub Scout is based on the arcade game *Frogger.* In this version of the game the player must guide an old grandmother to her house via a busy four lane highway and a river full of logs and turtles, all within an allotted time. Situated between the highway and the river is a narrow strip which offers refuge from danger. At the top of the screen is your final destination, the homes of the grannies.

Points are awarded for each lane of the highway crossed, and for each log or turtle encountered. A continual display of the number of grannies awaiting escort, the time remaining and the current score is shown at the bottom of the screen.

Akeila will be pleased if you succeed in guiding all three grannies home, an extra grannie being awarded for your efforts (as well as a Busy Beaver badge).

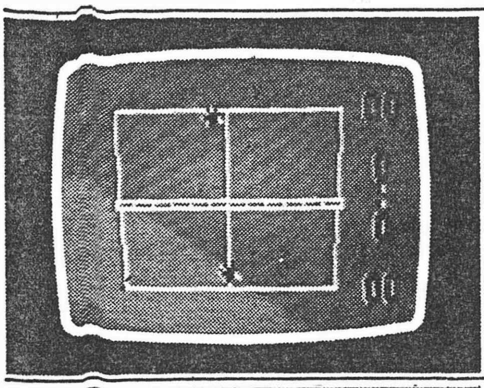
The continual vanishing and reappearing image of the cub and granny (in block graphics form), make precise location of the escorted granny somewhat difficult at times, especially when crossing the highway. The most successful method, although not scout-like, is to close your eyes and run for your life.

Written in both Basic and an Assembly language routine called from Basic, this game shows excellent use of the low resolution graphics facilities of the VZ-200. The absence of various levels of difficulty and the fairly poor use of sound distracts slightly from the game, however, the single degree of difficulty available is challenging, even to an experienced *Frogger* player, especially staying on the first row of extremely slippery logs.

The game can be played quite adequately with the use of the cursor control keys, however response was found to be frustratingly slow at times. The use of the joystick is preferred, especially when negotiating the extremely busy highway.

Although *Cub Scout* is not original in concept, it is regarded as one of the better arcade type games available for the unexpanded VZ-200. IT

GRAPHICS	***
SOUND	**
ORIGINALITY	*
LASTING INTEREST	***
OVERALL	***



GAME: Tennis

MACHINE: VZ-200 (24k)

JOYSTICK: Optional

SUPPLIER: Dick Smith

PRICE: $12.50

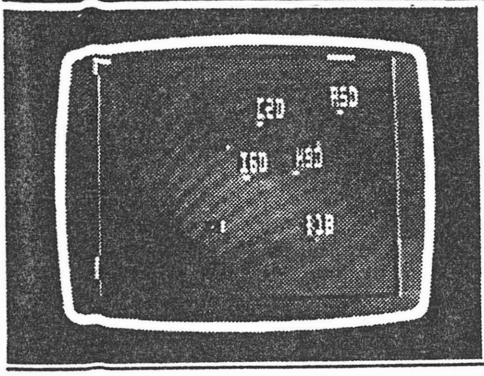
OVERALL: **

On loading this machine language program you are immediately presented with a demonstration of the game of *Tennis*. Follow the demonstration carefully as no other instructions are given.

One feature of the game is that the ball's height off the ground can be estimated by the position of the ball in relation to its shadow. This allows you to calculate the likelihood of the ball going over the side or back-lines on the full.

Scoring is the same as conventional tennis and players must change sides at the appropriate times.

Better games are available. **IT**



GAME: Air Traffic Controller

MACHINE: VZ-200 (24k)

JOYSTICK: No

SUPPLIER: Dick Smith

PRICE: $19.95

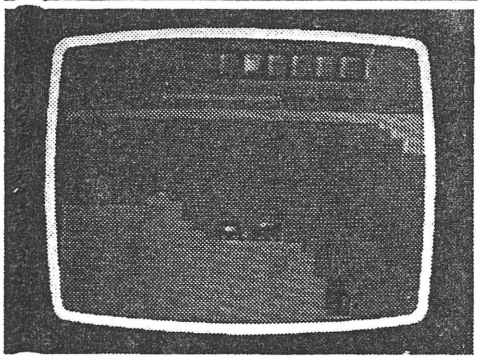
OVERALL: ****

The aim of the game is to provide the safe, orderly and expeditious flow of air traffic within controlled airspace.

You have 11 arriving aircraft and 9 departing aircraft, all of which you control. You must ensure that each aircraft lands at the correct airport, uses the correct runway and when departing are sent off your radar screen to the next sector at the correct radar exit location while maintaining a height of at least 7,000 feet. Simultaneously, at least five nautical miles horizontally or 1,000 feet vertical separation must be maintained between all aircraft under your control.

A great game for the VZ-200. A detailed booklet will be available. **IT**

PCG. Jan 85 p 64.



GAME: Defence Penetrator

MACHINE: VZ-200 (24k expanded)

JOYSTICK: No

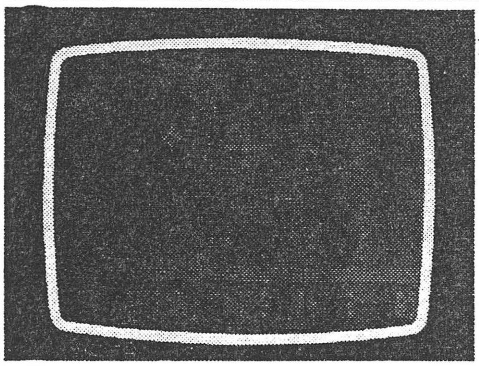
SUPPLIER: Dick Smith Electronics

PRICE: $12.50

OVERALL: ****

As the pilot of the Annihilator, the deadliest surface attack space craft known, your mission is to infiltrate enemy territory on a remote planet. Your craft is carrying the deadly Quacker 5000 air-to-surface super bombs.

Enemy defence will try to eliminate your craft with strategically placed auto-launch ballistic missiles and Skyhawk Destroyers. Rugged surface terrain and freak meteor storms hinder your progress. Your movement is also heavily restricted by a long winding subterranean passageway, together with massive missile launch bays which stretch hundreds of meters into the air. IT



GAME: Star Blaster

MACHINE: VZ-200 (24k expanded)

JOYSTICK: Optional

SUPPLIER: Dick Smith

PRICE: $12.50

OVERALL: ****

Your mission is to destroy the seemingly never ending array of enemy space craft

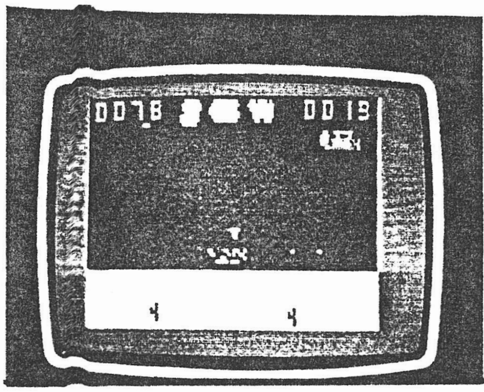
The enemy fleet consist of ships, tankers, freighters and troop carriers which must be destroyed. There are fighters that career straight towards you and will score a hit on your ship if you do not destroy them first.

Your Star Blaster laser is deadly accurate, however, the movement of enemy craft makes the task exceedingly difficult.

Using the joystick instead of the keyboard makes the game a little easier.

Excellent use is made of graphics, and the game is well recommended. IT

PCG Feb 85 p 76 1 of 1.



GAME: Planet Patrol

MACHINE: VZ-200 (24k expanded)

JOYSTICK: No

SUPPLIER: Dick Smith

PRICE: $12.50

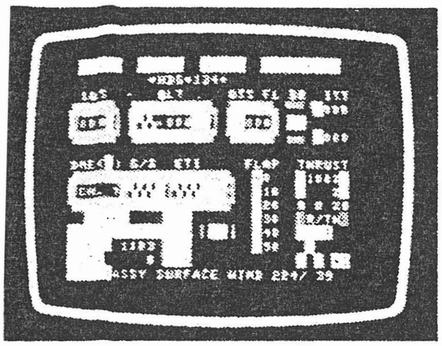
OVERALL: **

In this game you are in a moon-buggy negotiating the somewhat monotonous planet's surface, bounding over occasional craters and boulders. Moon-men also appear at intervals and must be destroyed by the appropriate use of the keys or joystick.

A space ship flying overhead drops bombs that must be destroyed, or dodged in your moon-buggy.

Points are scored according to the number of bombs and moon-men zapped and your ability in negotiating the craters and boulders.

Save your money, better games are available. IT



GAME: Learjet

MACHINE: VZ-200 (24k expanded)

JOYSTICK: No

SUPPLIER: Dick Smith

PRICE: $19.95

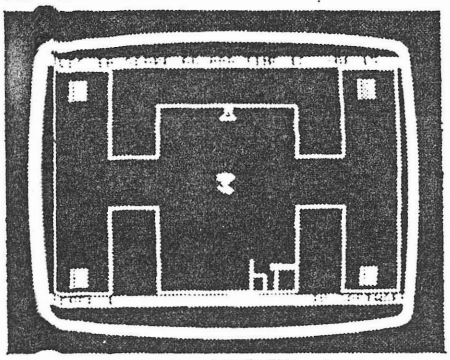
OVERALL: ****

The object of this flight simulator is to test your skills as a learjet pilot. Once you have chosen one of ten available routes, it is then up to you to successfully take-off, fly and land the aircraft using the least amount of fuel.

Before take-off you must set your instrument panel according to given information such as surface wind and direction and weight of fuel on board. Thrust settings and the best cruising altitude for economy are then determined according to comprehensive flight data supplied.

A basic knowledge of aviation is assumed, although not essential. IT

PCG Mar 85 p 76-77.



GAME: Asteroids
MACHINE: VZ-200 (expanded)
JOYSTICK: Optional
PUBLISHER: Dick Smith
SUPPLIER: Dick Smith
PRICE: $12.50

It's lonely out in space when you are the only one left to protect your planet from the continuous shower of meteors. Your mission is to destroy the meteors before they destroy you.

Fortunately you can rotate your ship in any direction, firing your deadly accurate lasers as you move about in space. However, as you shoot the larger meteors they break up and become lots of little ones, these smaller meteors being just as deadly as their parents.

If you become so trapped by meteors that you feel that your doom is near, you may escape by the use of your 'hyperspace' button, projecting you at random to another part of space. Use this button with extreme caution as you may be projected directly into the path of another meteor.

Armed with your trusty laser gun, the only safe way to survive is to continuously fire at all that comes your way, or even looks like coming your way.

You are given three lives before your doom is declared as final.

On-screen scoring gives you a continuous update of your game, bonus points being gained by shooting down enemy space craft that occasionally enter your air-space. Beware of these space craft, however, as they also fire lasers at you as deadly as your own.

The game is for one or two players, and using the joystick makes the game easier to play.

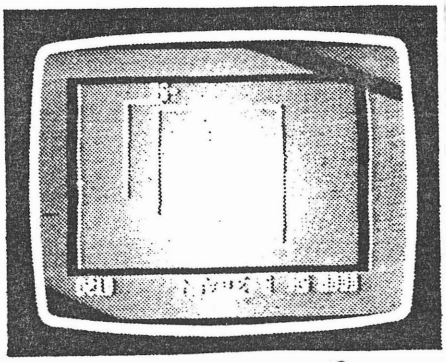
Written in assembly language called from Basic, the game makes excellent use of graphics. Better use could have been made, however of the sound features.

Asteroids is addictive to start, however I'm inclined to doubt whether the addiction will last all that long.

For $12.50 the game is nevertheless recommended as one of the better graphics games from Dick Smith for the VZ-200.

IT

GRAPHICS	****
SOUND	**
ORIGINALITY	****
LASTING INTEREST	***
OVERALL	****



GAME: Super Snake
MACHINE: VZ-200 (expanded)
JOYSTICK: No
PUBLISHER: Dick Smith
SUPPLIER: Dick Smith
PRICE: $12.50

You are a snake and in order to grow you must seek food which appears randomly on the screen.

To catch the food you must continue to move the snake around the playing area without touching either the walls or any part of your own tail.

You score points by eating the food as it appears. Each piece of food is worth a random value between 1 and 38. This value is added to your score and also to the length of your tail. As the length of your tail increases so it becomes more difficult to stay alive.

There are four levels of play with ten playing speeds within each level, giving a grand total of 40 levels of difficulty! The upper ten speeds, (champion level) are so fast that play is virtually impossible.

The levels of difficulty are selected by first pressing a letter (A-D) to set the level of play, followed by a number (0-9) to set the playing speed.

Using the control keys you then manoeuvre your snake around the screen, your tail becomes longer as you eat the food. If you do not eat the food withing a short period of time it disappears and re-appears in a new random location. In trying to make your catch you finally tie yourself in a knot due to the growing length of your tail, or you are forced into the walls due to lack of room.

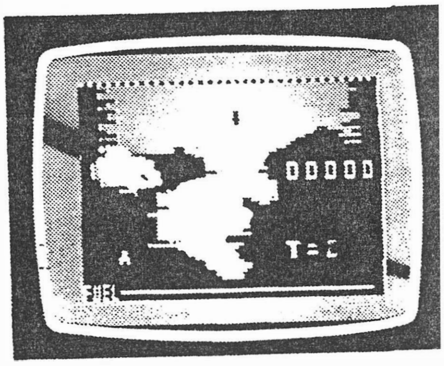
The screen shows a continuous score update of the game in play, together with the highest previous score gained for that level. All high scores are held in memory and are displayed in turn as each speed level is chosen.

Once you have mastered the control keys, *Super Snake* becomes a very challenging game to play, despite its simplicity.

Quick response and the ability to analyse the situation as you move around the screen make this game a real challenge and an excellent learning aid for children.

IT

GRAPHICS	****
SOUND	***
ORIGINALITY	**
LASTING INTEREST	*****
OVERALL	****



GAME: Lunar Lander
MACHINE: VZ-200 (expanded)
JOYSTICK: Optional
PUBLISHER: Dick Smith
SUPPLIER: Dick Smith
PRICE: $12.50

In this arcade-style game the object is t navigate your craft down a moon crat and onto the yellow landing pad pr vided without running out of fuel o crashing into the rocky lunar landscape or crater walls.

The crater is extremely rugged, making the task of landing your craft far more difficult than first appears. By the use o appropriate keys (or joystick) you mus guide your ship, increasing or decreasing thrust as you navigate past enemy lase beams.

You can protect yourself from these laser beams by turning on your 'force field', however, you must turn the force field off again before being able to land your ship. A bonus landing pad frequently appears in a small cave on the side o the crater, a welcome relief if you are running out of fuel.

Your ship's landing gear must also be in position before being able to land; any attempt to land on the pad without you landing gear in position will result in a crash landing. Unfortunately, you have no direct control over your landing gear, this being randomly set by the computer. If, when approaching the landing pad your gear is not in position (as shown by a blue bar at the base of your ship), you must hover (thrust setting 4) until the landing gear is set.

The screen gives you a continuous indication of your thrust setting (0-5) and your current score. Flashing bars indicate which of your turn settings (L or R) is set on. The absence of these bars means a straight descent. A moving scale at the bottom of the screen shows your fuel remaining.

Using a joystick is an advantage, since control of the craft is somewhat difficul when using the keyboard.

Fair use is made of the VZ-20 graphics, but better games ar available.

IT

GRAPHICS	****
SOUND	***
ORIGINALITY	***
LASTING INTEREST	***
OVERALL	***

PCG Apr 85 p.94, 97, 98, 99.

Log book and Morse course on VZ200

Two programs for CB and ham enthusiasts for the VZ200 microcomputer (unexpanded) have been developed by a new Tasmanian enterprise, Hi-com Programs.

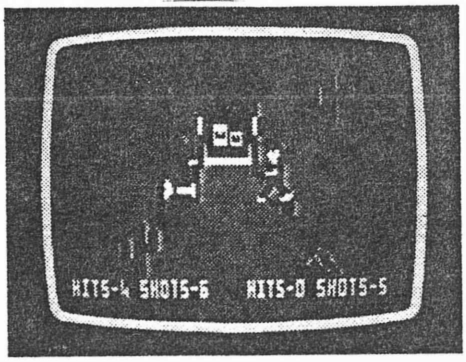
'Log Book' takes advantage of VZ200 command of INPUT# and PRINT#, which enable you to load and retrieve file data from the tape while the program is running.

Included in this 'log book' package is a similar program that uses DATA commands to load and retrieve file data.

'Morse Code' is aimed at an operator studying for a novice amateur licence. As well as teaching the Morse code, the operator can be drilled in single letters, single words or full sentences; this can be from letters to code, or vice versa. This program is claimed to be based on sound educational ideas and gives some assistance when errors are made.

Program tapes are available at $6 for the log book package; $6 for the Morse code; $10 for both programs.

For further information contact **J. Hirst, Hi-com Programs, RSD 170, Exeter, Tas 7251. (003)94-4003.**



GAME: Duel

MACHINE: VZ-200/300 (unexpanded)

PUBLISHER: Dick Smith

SUPPLIER: Dick Smith Electronics

PRICE: $13.95

OVERALL: **

Duel consists of two games on a single tape, both games being player against player. The first program, *Ace of Aces,* is a game where you have to hit and destroy your opponent's plane. A total of 15 hits is required to cause total destruction.

After 15 hits your plane goes up in a puff of smoke, the program then returns for a second duel. Do not think that you can hide behind the clouds as your opponent's guns are just as deadly even when you're not in direct view.

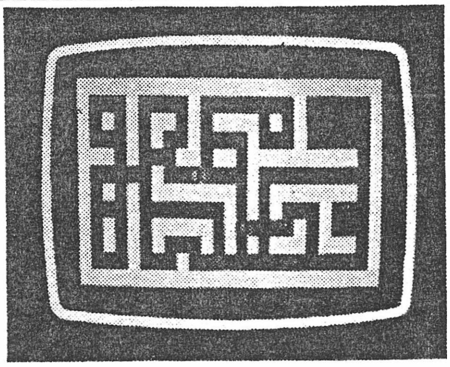
Poor use is made of sound. The only sound is a beep each time you fire your gun.

In the second program, *Gunfighter,* you and your opponent are set for a duel, each armed with a six-shooter.

Both programs make fair use of graphics, however, the poor use of sound does distract somewhat from the games.

Although both games can be played using the keyboard, the use of joysticks is preferred as the keyboard does become crowded with a total of ten keys being used between two players. IT

PCG Oct 85 2(7)
p. 68-69



PCG. Nov 85. 2(8)
p 70-71.

GAME: Attack of the Killer Tomatoes
MACHINE: VZ-200/300 (unexpanded)
PUBLISHER: Dick Smith
SUPPLIER: Dick Smith Electronics
PRICE: $13.95

In *Attack of the Killer Tomatoes* you are trapped in a maze with up to five extremely vicious vegetables. If they catch you they will kill you. All, however, is not lost. You can destroy the killer tomatoes by digging holes with your shovel and trying to lure the tomatoes into the holes.

Once they fall into a hole they are momentarily trapped; to kill the tomato you must then bury it.

Remember that even though the killer tomatoes have very poor eyesight and can't see your holes, they are big. You need a large hole to trap them and even then you have to be quick to fill the hole before they can escape and chase you.

Tomatoes may be stupid but they will help one another. If one is trapped in a hole, another will help it out. So be wary of tomatoes which travel in a convoy.

In each game you have two spare lives. If you take too long finishing a game the tomatoes will go wild so it is advisable to bury the tomatoes as quickly as possible.

Caution is also required as the tomatoes can merge and divide again as they chase toward you.

Although the game can be played with the keyboard, the use of joysticks is recommended. With completeabsence of sound, and the poor use of graphics, better games are available for the VZ-200.

The game keeps a tally of the highest score. If you wish to save the highest score for later retrieval, simply press 'E' for exit before starting the next game, this command will send you from machine language to Basic. The program, however, will still remain in RAM. Without entering any other commands, CSAVE the program on a blank tape. IT

GRAPHICS	**
SOUND	N/A
ORIGINALITY	**
LASTING INTEREST	**
OVERALL	**

Keyboard, Tower of Hanoi and Block Puzzler

VZ 200 For ages 5-8

The three programs evaluated below were trialled with children in Years 2/3 and Years 5/6, using an unexpanded VZ-200 microcomputer.

Keyboard

This program introduces keyboard manipulation to child and adult alike, through a game situation. The monitor displays a key and the pupil must press the corresponding key upon his keyboard within an allotted time limit. At the conclusion of each game a score out of twenty is registered. There are six skill levels with the time allowed for each response diminishing at each increased skill level.

Cassette $8.00
VsoftwareZ

Tower Of Hanoi

The aim of the program is to shift a group of disks from one pile to another. The shifted disks must then be rearranged in order from smallest to largest in their new location. Arrow keys control all movements.

The player has a choice of three skill levels : 3 disks, 5 disks and 7 disks which need to be reassembled within a minimum number of moves. As well as the challenge of solving the problem within a minimum number of moves, a timer makes the game a race against the clock.

The documentation and on-screen instructions are both clear and concise.

Cassette $8.00
VsoftwareZ

Block Puzzler

This is another logic and mathematical problem solving program. The aim of this game is to rearrange a set of randomly dispersed letters into a matching sequence. This sequence has to be arranged alphabetically. The program only allows children to complete the task within a minimum number of moves or within 10 minutes duration.

Block Puzzler is supposed to suit Years 4-7, however I would recommend its use only with mathematically gifted children in the lower primary/infants level. It would be more suitable for use with children in the upper primary and early secondary school years.

Reviewer *Rhys McGregor*

Cassette $10.00
VsoftwareZ

VsoftwareZ
39 Agnes Street
Toowong, Qld 4066
Tel: (07) 371 3707

Classroom Computing 2(6) Nov 85 p. 31.
(Publ. by Ashton Scholastic)

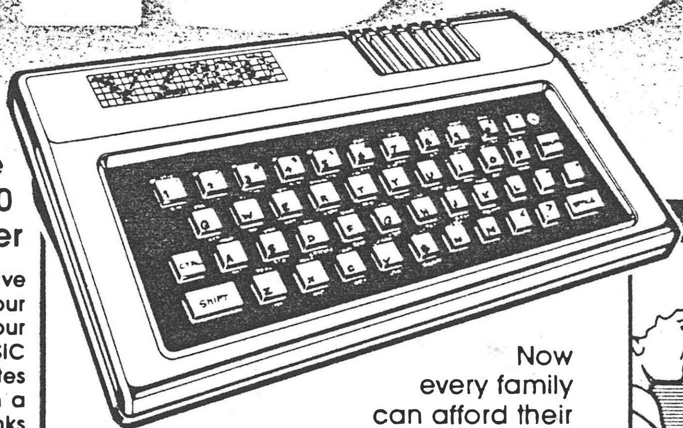
SOFTWARE ADVERTISEMENTS

A 15 page compilation of ads. for a variety of software, services, User groups etc. (12)

AMAZING BREAKTHROUGH!

Colour Computer

$199



The incredible DICK SMITH VZ 200 Personal Colour Computer

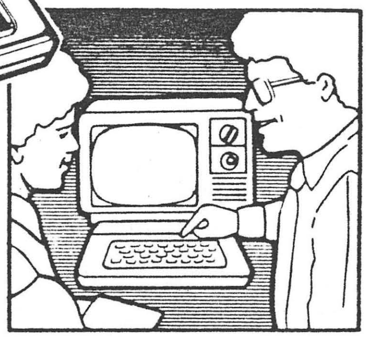
Here it is at last — the breakthrough you've been waiting for! A personal colour computer with all the right features: colour graphics, sound, standard Microsoft BASIC for easy programming, a whopping 8K bytes of RAM memory, the ability to work with a standard TV set, and much more. Yet thanks to modern electronics and our buying power, the Dick Smith VZ 200 will cost you only $199 — far less than any comparable computer! There'll never be a better time to invest in your family's future..

Yes, for just $199, the Dick Smith VZ 200 gives you amazing computing power — far more than many machines two, three or even four times the price. Now you can find out what computers are all about. The kids can use it with their school work. It can keep track of your home budget. It can even help you in your business!

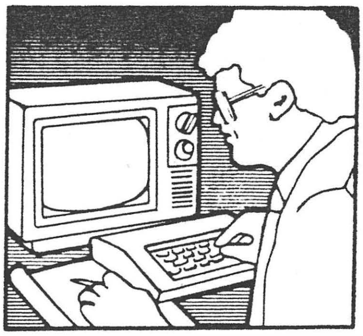
Still not convinced? Try our exclusive 7 day money back satisfaction guarantee.

Buy the Dick Smith VZ 200 Colour Computer and try it in your home for up to 7 days. If you're not absolutely delighted, you can return it in original condition and packaging for a full refund.

You'll owe nothing — not even an explanation!



Now every family can afford their own personal computer



- **Simple and safe to use**
  Operates from low voltage via a mains adaptor Absolutely safe — even for children.
- **Works with any normal TV set**
  It simply plugs in, no need to buy an expensive monitor
- **Uses a normal cassette recorder**
  No need to buy a high cost computer type recorder.
- **Easy to read manuals, Demo cassette**
  When you buy the VZ 200, you get not one but two manuals, a User's Manual and a BASIC Manual, plus a Demonstration Cassette

That's the incredible DICK SMITH VZ 200

Cat. X-7200

ONLY AVAILABLE FROM

DICK SMITH ELECTRONICS

NSW • Auburn 648 0558 • Bankstown Sq. 707 4888 • Blakehurst 546 7744 • Bondi Jct 387 1444 • Broadway 211 3777 • Brookvale 93 0441 • Chullora 642 8922 • Gore Hill 439 5311 • Gosford 25 0235 • Liverpool 600 9888 • Newcastle (Tighes Hill) 61 1896 • North Ryde 88 3855 • Parramatta 689 2188 • Penrith 323 400 • Sydney (Bridge St) 27 5051 • Sydney (York St) 267 9111 • Tamworth 66 1961 • Wollongong 28 3800 **ACT** • Fyshwick 80 4944 **VIC** • Coburg 383 4455 • Frankston 783 9144 • Geelong 78 6766 • Melbourne 67 9834 • Richmond 428 1614 • Springvale 547 0522 **QLD** • Brisbane 229 9377 • Buranda 391 6233 • Chermside 59 6255 • Toowoomba 38 4300 • Townsville 72 5722 **SA** • Adelaide 212 1962 • Darlington 298 8977 • Enfield 260 6088 **WA** • Perth (William St) 328 6944 • Perth (Hay St) 321 4357 • Cannington 451 8666 **TAS** • Hobart 31 0800

A 527 JL

ORDER BY PHONE!
Just phone us on (02) 888 2105 and quote your Bankcard No Your VZ-200 will be on its way the same day!!!

A.P.C. Jun/Jul 83 4(5/6)

APC Aug 83 4(4) p 70.

YC Aug 83. p 60.

VZ-200

DEFENCE PENETRATOR

Can you destroy strategically placed missile bases or will they SCRAMB our system?

Now you are the Pilot of the VZ-200 Annihilator, the deadliest surf attach spacecraft known. Your mission is to infiltrate enemy territory on rem planetoids carrying the deadly QUACKER 5000 air to surface super bom and your own crafts high output intercept to destroy torpedoes. Destroy u destroyed!

Enemy defences will try to eliminate you with carefully stationed a launch ballistic missiles and Skyhawk Destroyers. Rugged surface terrain a feak meteor storms will hinder you as you rain down fiery devastation u their surface and underground nuclear reactors and supply depots. W NERVES of STEEL you must try to recover your steadily diminishing f supply in flight and reach the climax of your mission!

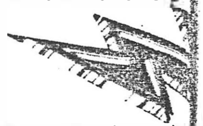
Can you cross the 2000 KM of swooping scenarios?

Can you reach the enemy COMMAND BASE and smash it to atoms?

Danger awaits you with super ARCADE ACTION!

DEFENCE PENETRATOR is based on one of the most popular arc favourites of all time with FAST ANIMATED HIGH RESOLUTION COLO GRAPHICS and SOUND EFFECTS. Written in machine code for supe quality.

16K TAPE $12.50



WIN!!

With every $20.00 you spend, you will receive a chance to win a priz Cosmic Software's "Totally awesome & mysterious Lucky Num Draw!" (All prizes worth not less than $100).

RUSH YOUR ORDER NOW

HOW TO ORDER:
Order the products you want on our COSMIC HOTLINE! Ph: (02) 661 4075. Use your Bankcard. Or order direct through our superfast Mail Order Department! See order form. If you do not want to cut it out then just copy out the main details! All orders are sent out to you within 24 hours of receiving your order by CERTIFIED PRIORITY PAID MAIL. RUSH YOUR ORDER NOW

APC. Oct 83 4(10)

Post to: COSMIC SOFTWARE
G.P.O. Box 3494, SYDNEY N.S.W. 2001
Phone: (02) 661 4075

NAME ....

ADDRESS ....

.... P/CODE ....

Enclosed$ .... Bankcard/Cheque/ Money Order

Expiry Date: ....

Bankcard Number ....

Signature : ....

ITEM	QTY	$ PRICE EACH c	$ AMOUN

ALL PRICES INCLUDE POSTAGE! TOTAL

APC Oct 83 4(10)

VZ200 Games Featuring Hi-Res Colour Graphics and Fantastic Sounds.

MORGOTH

Prepare yourself for medieval adventure in MORGOTH. Now you are pitted against fire-balls, giant spiders, ghosts and ghouls with only your bow and arrows for protection. Can you do battle in the underground caverns, ancient castle or the mirky forest? Find out. Morgoth features fantastic sound, a tantastic challenge and hi-res colour graphics!

VZ200 TAPE 16K, $12.50

BOSKONE ALERT

The ULTIMATE experience in space combat. Battle "intelligent" escort fighters, drone kill pods, and attempt to destroy the 9 Deathstars. Complete with over 100 screens of space "map" to explore and a scanner to guide your craft with.

Move 8 directionally (all terrain scrolls 8 directionally) and fight in the time-space twisted Vortex field or the moving asteroid belt! Fast moving animated machine language action beyond any arcade game ever written! With sound! Includes detailed battle manual!

VZ200 TAPE 16K, $12.50

RALLY RACER

Beware! Mad Morgan, Crazy Harry and his hoodlums are on the prowl in a fantastic maze trying to track you down. Is your super charger fast enough! Your car remains in the centre of the screen while objects move around it! Your fuel is limited and you have to knock down 10 flags, but don't despair. A grid scanner to your right indicates the position of your cars and your foes!

VZ200 TAPE 16K, $12.50

DEFENCE PENETRATOR

Can you place strategically placed missile bases or will they SCRAMBLE our system?

Now as pilot of the Z-80 Anbnihilator your mission is to infiltrate enemy territory carrying deadly QUAKER 5000 space-to-surface super bombs and your own crafts high high output intercept to destroy torpedoes.

Enemy defences will try to eliminate you with auto-launch ballistic missiles and installations. Beware of rugged surface terrain, your diminishing fuel supply and meteor storms!

Can you cross the 2000 km of swooping scenarios?

Can you reach the enemy COMMAND BASE and smash it to atoms? Written machine language with super smooth hi-res colour graphics and sound!

VZ200 TAPE 16K, $12.50

VZ Monitor

Monitor Dis-Assemble (Z80 Code). Single Steps Programmes and many other features.
The BEST Monitor Available!
VZ200 16K Tape $19.95

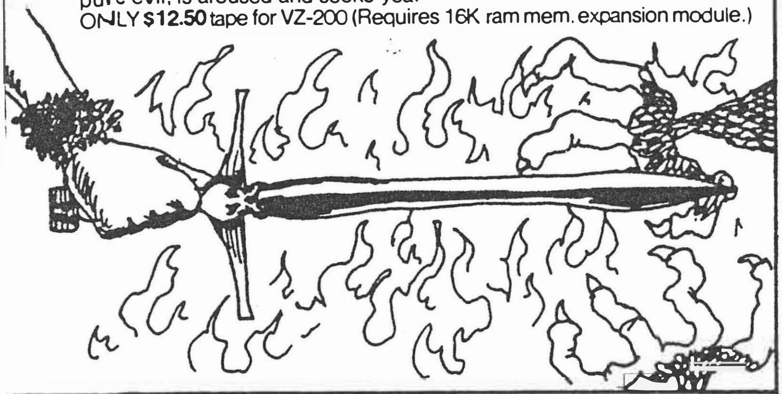
TO ORDER
TURN TO PAGE 82.

YC Nov. 83 p 73.

VZ-200

MORGOTH

MORGOTH Morgoth is a multi-roomed arcade adventure which combines the fast paced skill and action of arcade games with the heroic deeds of an ancient world. It's fully animated, smooth Hi-Res colour graphics and sound effects highlight the CAVERNS OF MORIA, SMEAGORS KEEP & MIRKWOOD FOREST where you are pitted against a variety of monsters with only your magic bow and arrows for protection. Morgoth is a unique action packed adventure allowing you to wander through the enchanted dominion of Morgoth and collect the lost treasures of KAZARD KALLAHAN. But, beware! You must escape before the satanic MORGOTH, being of pure evil, is aroused and seeks yea!
ONLY **$12.50** tape for VZ-200 (Requires 16K ram mem. expansion module.)



BOSKone ALERT

BOSKONE ALERT The ULTIMATE experience in space combat. Battle "Intelligent" escort fighters, Drone kill pods and attempt to destroy the 9 Deathstar Stations. Move 8 directionally (all terrain scrolls 8 directionally) and fight in the time-space twisted Vortex field or the moving asteroid belt! Fast moving Hi-Res colour animated machine language action beyond any arcade game ever written! With sound!!
ONLY **$12.50** Tape for VZ-200.(Requires 16K Ram mem. expansion module).

Road Warrior

ROAD WARRIOR Beware! Mad Morgan, Crazy Harry and his hoodlums are on the prowl in a fantastic (32 screens) of maze trying to track you down. Is your super charged car fast enough! Your car remains in the centre of the screen while the background moves around it! Your fuel is limited and you have to recover 10 fuel tanks marked by flags, but don't despair. Your car is armed with a smoke screen and scanner! Fast Hi-Res colour graphics and sound! Very fast ACTION!!!
ONLY **$12.50** tape for VZ-200 (Requires 16K ram mem. expansion module).

DEFENCE PENETRATOR

Can you destroy strategically placed missile bases or will they SCRAMBLE our system?

Now you are the Pilot of the VZ-200 Annihilator, the deadliest surface attach spacecraft known. Your mission is to infiltrate enemy territory on remote planetoids carrying the deadly QUACKER 5000 air to surface super bombs and your own crafts high output intercept to destroy torpedoes. Destroy until destroyed!

Enemy defences will try to eliminate you with carefully stationed auto-launch ballistic missiles and Skyhawk Destroyers. Rugged surface terrain and feak meteor storms will hinder you as you rain down fiery devastation upon their surface and underground nuclear reactors and supply depots. With NERVES of STEEL you must try to recover your steadily diminishing fuel supply in flight and reach the climax of your mission!

Can you cross the 2000 KM of swooping scenarios?

Can you reach the enemy COMMAND BASE and smash it to atoms?

Danger awaits you with super ARCADE ACTION!

DEFENCE PENETRATOR is based on one of the most popular arcade favourites of all time with FAST ANIMATED HIGH RESOLUTION COLOUR GRAPHICS and SOUND EFFECTS. Written in machine code for superior quality.

16K TAPE $12.50

MAIL LIST This easy to use program will enable your VZ-200 to create and maintain a file of up to 200 names, addresses & phone numbers.

Ideal for: —Mailing lists for small businesses
—Membership list for clubs, associations, churches, schools societies etc.

ONLY **$12.50** Tape for VZ-200.(Requires 16K Ram memory expansion module and Printer).

VZ-MONITOR VZ-MONITOR provides the user with a system level interface which is a must for any application outside BASIC programming. A monitor allows the user to display and change memory contents, move memory to and from cassette, as well as fill, search and compare memory. Disassembles all ZILOG Z-80 codes!
ONLY **$19.95** Tape for VZ-200.(Requires 16K memory expansion module).

MOST OF THE ABOVE VZ-200 SOFTWARE IS AVAILABLE FROM YOUR LOCAL DICK SMITH ELECTRONICS STORE

RUSH YOUR ORDER NOW

HOW TO ORDER. Order the products you want on our COSMIC HOTLINE! Ph: (02) 661 4075. Use your Bankcard. Or order direct through our superfast Mail Order Department! See order form. If you do not want to cut it out then just copy out the main details. All orders are sent out to you within 24 hours of receiving your order by CERTIFIED PRIORITY PAID MAIL. RUSH YOUR ORDER NOW.

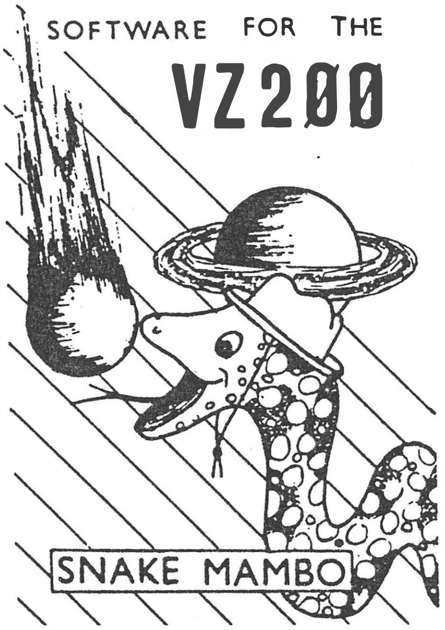
Post to: COSMIC SOFTWARE
G.P.O. Box 3494, SYDNEY N.S.W. 2001
Phone: (02) 661 4075

NAME
ADDRESS
P/CODE
Enclosed$ Bankcard/Cheque/Money Order
Expiry Date
Bankcard Number
Signature

ITEM	QTY	$ PRICE EACH c	$ AMOUNT

ALL PRICES INCLUDE POSTAGE TOTAL

VZ-200 Software: The author of 'Acne Attack - the game that set everyone laughing...', described in the October 8-9 1983 edition of the *Weekend Australian*, presents 'Snake Mambo'. 'Snake Mambo' is a dynamic chase game utilising excellent graphics and sound effects. Guide Mambo, the galactic snake, to various energy pills to help him regain his strength and your score. Mambo needs help and only you can save him! Program uses joystick or keyboard. Computer verified tape with full instructions, $10. Mail order to William Dickinson, 'Mount Pleasant', Bannockburn 3331. Prompt return guaranteed.



YC Apr 84.

VZ 200 SOFTWARE

An increasing range of quality software.
Rapid turnaround of orders and enquiries.
Realistic prices and helpful user support.

SUPER INVADERS: 10 levels, fast action and great sound effects.......... **$10.00**

PAKMAN: 4 different mazes, bonus symbols and great sound effects........ **$10.00**

GRAPHICS PACKAGE: Character designer, drawing commands and demonstration.................. **$10.00**

VZ DEBUG: Disassembler and monitor combined, over 20 commands **$20.00**

BLANK DATA CASSETTES: Top quality reliable loading everytime......... C12 **$1.85**

.... C24 **$2.25**

HANDLING CHARGES: Per order.................. **$1.50**

Send S.A.E. for further information or your order to:

R.S. MILLER
8 Mulga Street, Altona 3018
PHONE ENQUIRIES:
J. Hawley (03) 367 1469

Dec 83 4(12)

VZ200 (expanded): Adventure program, 'Castle of Elgior'. Send $10 to A Majajas, 99 Wyrallah Road, Lismore 2480.

YC Apr. 84 p 160.

VZ 200 SOFTWARE: Send SAE to R. Miller, 8 Mulga St, Altona 3018.

Hawley-Miller Software

13/5/86. Mr J. Hawley, Denton Av, St Albans V.
367-1469.
Monitor based on Les Bells series on Assembler in YC.
No longer supports software, has moved to AT&T
No longer concerned with copyright
Doesn't have copies of listings/software.

VZ 200 SOFTWARE

An increasing range of quality software. Rapid turnaround of orders and enquiries. Realistic prices and helpful user support.

SUPER INVADERS: 10 levels, fast action and great sound effects.	$10.00
PAKMAN. 4 different mazes, bonus symbols and great sound effects.	$10.00
GRAPHICS PACKAGE. Character designer, drawing commands and demonstration.	$10.00
VZ DEBUG: Disassembler and monitor combined, over 20 commands	$20.00
BIORHYTHM: Chart your way to success Printer or hi-res output	$10.00
MORSE-CODE: Learn Morse the easy way without having to buy a practice kit	$10.00
Handling Charges. Per Order	$1.50

Send S.A.E. for further information or your order to —
R.S. Miller, 2 Guinane Ave. Hoppers Crossing 3030
Phone Enquiries: J. Hawley (03) 367 1469

YC Jul 84 p 130

Aug 84 5(8)

YC Nov 84. p. 168.

YC Jul 84 p 129

YC Nov 84. p. 173.

FOR SALE: VZ 200 Extra listable commands. Auto, Trace, Delete, On goto, Onerror, String$, Defdbl, etc. Cassette and Information. $10-$15 G. Lehmann, 6 Midway Rd, Elizabeth East SA 5112.

128 — ETI December 1984

VZ 200 VZ 200 VZ 200

MONITOR/DEBUGGER FOR VZ 200

Save, load M/L tapes
Set break points, set flag reg's
Mem. dump's to printer and more
$14.95 only
Renumber, merge basic programs
(M/L tape) $12.50
(Both programs need mem. expansion)
by
W. OBRIST
P.O. Box 56 St. Kilda 3182

VZ 200 VZ 200 VZ 200

APC Dec 84 5(12)
p. 213.

FOR SALE: VZ200/300 UNIQUE and first class software, monitor/debugger $14.95, extended BASIC $12.50, array utility $14.95, protect utility $14.95 and more. W. Obrist, 50 Cobham Ave, West Ryde, NSW 2114.

ETI May 1986 — 63

5/8/86. W O'B has upgraded software to be relocatible in various VZ's.
41 O'Hara St,
Marrickville NSW. 2204.

VZ-200 SOFTWARE

WORD PROCESSOR. A simple menu driven programme outputs multiple copies to printer in letter format or text only. Features insert, delete, alter, tape save & retrieve. Requires 16K exp. please indicate 40 or 80 column printer.

DATA MASTER. Data base for cataloguing books, record collections etc. Full editing, fast sort & search routines. Outputs to printer all or part of files. Requires 16K exp.

ADDRESS BOOK. Stores up to 200 names, addresses and telephone numbers. Outputs to printer and saves on tape. Field search and fast sort routines. Requires 16K exp.

$8.00 each P&P inc. or all 3 for only $20.00

L. DAWSON
BOX 718F G.P.O. MELBOURNE 3001

APC Dec 84 5(12)
p. 214.

VZ-300/200

Asteroids
Dick Smith
Cassette $13.95

Backgammon
Dick Smith
Cassette $13.95

Blackjack
Dick Smith
Cassette $9.50

Checkers
Dick Smith
Cassette $13.95

Chess
Dick Smith
Cassette $13.95

Circus
Dick Smith
Cassette $9.50

Dracula Castle
Dick Smith
Cassette $13.95

Duel
Dick Smith
Cassette $13.95

Dynasty Derby
Dick Smith
Cassette $9.50

Golf/Tennis
Dick Smith
Cassette $9.50

Hangman
Dick Smith
Cassette $9.50

Hoppy
Dick Smith
Cassette $13.95

Horse Racing
Dick Smith
Cassette $9.50

Killer Tomato
Dick Smith
Cassette $13.95

Knights/Dragons
Dick Smith
Cassette $13.95

Ladder Challenge
Dick Smith
Cassette $13.95

Matchbox
Dick Smith
Cassette $9.50

Othello
Dick Smith
Cassette $13.95

Planet Patrol
Dick Smith
Cassette $13.95

Poker
Dick Smith
Cassette $9.50

Slot Machine
Dick Smith
Cassette $9.50

Star Blaster
Dick Smith
Cassette $13.95

Super Snake
Dick Smith
Cassette $13.95

Tennis
Dick Smith
Cassette $13.95

VZ Ghost Hunt
Dick Smith
Cassette $13.95

VZ Invaders
Dick Smith
Cassette $9.50

VZ Panic
Dick Smith
Cassette $13.95

GENERAL BUSINESS

Mailing List
Dick Smith
Cassette $13.95

Statistics 1
Dick Smith
Cassette $13.95

Statistics 2
Dick Smith
Cassette $13.95

EDUCATION

Flashcard 1+2
Dick Smith
Cassette $13.95

Flashcard 3+4
Dick Smith
Cassette $13.95

Intro To BASIC
Dick Smith
Cassette $13.95

Matrix
Dick Smith
Cassette $13.95

Metric Spycatcher
Dick Smith
Cassette $13.95

Music Writer
Dick Smith
Cassette $13.95

Speed Reading
Dick Smith
Cassette $13.95

Spellomatic 1+2
Dick Smith
Cassette $13.95

Spellomatic 3+4
Dick Smith
Cassette $13.95

Typing Teacher
Dick Smith
Cassette $13.95

Whizkid Spycatcher
Dick Smith
Cassette $13.95

GENERAL UTILITIES

Disassembler
Dick Smith
Cassette $13.95

Editor Assembler
Dick Smith
Cassette $22.95

Hex Utilities
Dick Smith
Cassette $13.95

MISC

Biorhythm
Dick Smith
Cassette $9.50

YC Jun. 85. p 94.

YC Mar 85 p 126.

YC Sep 85 p 160.

APC Oct 85 6(10) p 210.

APC May 86 7(5) p 157

YC Nov 84. p 173.

VZ200 Tenpin Bowling Program (expanded): Test your bowling skill with this computer simulation. Send $11 to GJ McCleary, 1 Grey Street, Emu Plains 2750.

YC Jul 84 p 129.

VZ 300/200
FREE CATALOGUE
All original games
Commodore 64
Talking Sam
Your Computer Friend
$22.45
Disk or tape
Write to:
Gary McCleary Software
P.O. Box 24
Emu Plains 2750 NSW

Page 140 Australian Personal Computer

Jan 86 7(1)

FOR SALE: VZ200 Quality Software. ATC Learjet, EDASM, Word Processor, Panik, Ladder Challenge, Star Blaster, Defence Penetrator etc. Special offer! 21 programs $38 + $3 p&p. (02)982-5965.

ETI Dec. 85 p 121.

FOR SALE: Z200/300 PROGRAM COPIER. Copies all m/c games etc. Also transfer from tape to disk — $12 tape of $15 disk. N. Sarafoudis, PO Box 31, Huntingdale, Vic 3166. (03)551-6381 ah.

ETI Jan 86 p. 97

FOR SALE: VZ200 SOFTWARE. 25 exciting games on one cassette for the incredible price of $20 (inc p&p). Send cheque or money order to Lee Tait, PO Box 13, Auburn, SA, 5451 for prompt delivery.

96 — ETI March 1986

VZ200 Software — Program Copier
Will copy word processor, editor assembler, planet patrol, chess, tennis and all the other VZ200 software. Only $10 inc. P+P. Send cheque or money order to Nick Sarafoudis, PO Box 31, Huntingdale 3167.

03 551-6381
Contacted 14/5/86

YC Jul 85 p 143.

Wanted
To buy, sell or swap programs for VIC-20, Commodore 64, System 80 and VZ200 machines — good prices. Contact Brett Tollis, PO Box 584, Port Macquarie 2444.

YC Jun 86 p. 131.

VZ200/300 Software
Address book: can handle up to 350 files, bar charts mailing labels and so on. Shopping List: stores products and prices; offers list and data menu's; holds over 400 files, saves list, prints list and so on. Birthday Reminder: handles up to 500 files and calculates age. All save to disk or tape. All come are supplied with more utilities on one disc ($15) or tape ($12). For more information, contact C. Rhodes, 4/6 Eridunda Court, Lilydale 3140; (03) 735 1663.

YC Sep. 86 p 129.

FOR SALE: VZ200/300 software. Back up your valuable tapes. Makes copy of any VZ200/300 tape. Send $12 to E. Sundstrup, 29 Hunter St, Macedon, Vic 3440.

FOR SALE: VZ200/300 EXPANSION UNITS. 32K RAM for VZ200 only $80 and 18K RAM for VZ300 only $70. Ring David, on (07)209-8478 after hours.

ETI October 1986 — 49

PCG Nov 85 2(8)
p 35

APC Jan 86 7(1)
p 92.

APC Jul 86 7(7) p 199.
Lysco ph no. (097) 26 3009.

VZ-200/300 Computer Club	VZ-200/300	—	VZ-200/300 Club 24 Albert Street, Goodna Qld 4300 (07) 288 3045
VZ South Pacific User Group	VZ-200/300	—	John D'Alton 39 Agnes Street, Toowong Qld 4066 (07) 371 3707

APC Jan 86 7(1): 134.

13\5\86. Club folded in Jun85 – 60/65 members.
8 newsletters "VDU"
photocopied 12-35 pg. each.
9\8\86 Moved from address ~ 1 month ago.

Victorian VZ-200 User Group, Luigi Chiodo, 24 Don St, Reservoir 3073, (03) 460 3770.

YC Jan 86 p 190.

Gordon Browell

VZ200 Users Club, 7 Abbott Cres, Malak, Darwin 5793. (089) 27-2830.

ETI August 1984 — 135

● A new group for users of the VZ 200 computer has been formed in Victoria. Plans are to publish a newsletter every six weeks and to set up a software library for the use of members.

Subscription to the group of $10 per year and further information is available from Luigi Chiodo, 24 Don St, Reservoir, Vic 3073.

ELECTRONICS Australia, January, 1984 129

VZ USERS: Newsletter/mini magazine for VZ200/300 users. Send S.A.E. to 'VZ USER' P.O. Box 154, Dural 2158, for more details.

120 ELECTRONICS Australia, August 1986

CLUB CALL

Announcing the **VZ200/300 User Group** which hails from the postal address PO Box 316, St Kilda, Vic 3182. Those interested in joining could contact Scott Le Brun.

ETI September 1986 — 61

Details of the Ad Lib VeeZed Micro Club may be obtained by writing to Gordon Browell, Ad Lib VeeZed Micro Club, 13 Brookes Street, Biggenden Qld 4621.

APC Jul86 7(7) p. ~~146~~ 169.

The **Ad Lib VeeZed Micro Club** advises change of address to 13 Brookes St, Biggenden, Qld 4621. The club publishes a newsletter of particular interest to beginners, called *"Micro Magic"*.

46 — ETI October 1986

A new VZ-200/300 User Group has been formed. Interested readers should write to: VZ-200/300 User Group, PO Box 316, St Kilda Vic 3182.

The Ad Lib VeeZed Micro Club, previously based in Darwin, is now operating from Biggenden in Queensland. For more information contact: Ad Lib VeeZed Micro Club, 13 Brookes Street, Biggenden Qld 4621.

Australian Personal Computer Page 181

APC Oct 86 7(10)

Quickwrite Wordprocessor
Vsoftwarez
Phone: (07) 371 3707
Price: $40
For all those VZ 200 and 300 personal computer owners, you haven't been forgotten — there's a new wordprocessor designed for expanded VZ200 and 300 machines. The software features automatic periodic saving of text while in typing mode, if required, printing font changes within the data, fast disksaving and loading of document text, accommodation for wide printers up to 255 columns. You can choose either printer or plotter, four justify and ragged modes for printing, and labeling of disks with date, code and other means of identification. This is all in addition to the normal editing facilities available to a word processor.

YC Jun 87 p. 128.

VZ200 VZ300
AT LAST
A special book of Programme Listings, Basic and machine code, hardware modifications and more
Is a must for all VZ200/VZ300 owners and users

VPROGRAMMES — VHINTS — VHARDWAREZ

Postage included in Australia **A$18.50**

We also run Le'VZ 200/300 OOP user group. Newsletter $1.00 each

SUPERB SMALL BUSINESS EDUCATIONAL & UTILITY SOFTWARE

Le'VZ D'Base V1.8	**$98**
Le'VZ Statement V2.0	**$185**
Cash Book Ledger	**$65**
Edudisk — 8 programmes	**$50**
Meat Pies V2	**$20**
Maths Countdown	**$20**
Copy-protect	**$35**
Disk Guard Dissables Dcopy	**$60**
Monitor Debugger	**$15**
Extended Basic (with 23 extra commas)	**$15**
Load XX80 Files	**$8**

and many more — send large S.A.S.E. for VLISTZ
BANDCARD AND VISACARD WELCOME

VSOFTWAREZ
39 Agnes St., Toowong Qld 4066 Australia
Ph (07) 371 3707

Aug. 86 7(8)

VZ-300
Software — many titles. Write for a free comprehensive list. Scott Le Brun, 5 Cameron Crescent, Wantirna Vic 3152.

YC Oct 87 p 159

Wanting to Join
VZ-300 Club. Please send details on joining. P. Miller, PO BOX 174, Forbes 2871 NSW.

VZ-300
Wanted — education programs and foolproof Yatzhee program, especially. P Miller, PO BOX 174, Forbes 2871 NSW.

YC Nov 87 p 159.

VZ200/300 INFORMATION. The largest user group in the South Pacific area. Le VZ 200/300 OOP. Send S.A.S.E. to Mr. D'Alton, 39 Agnes St. Toowong 4066, Qld. Ph. (07) 371-3707.

VZ200/300 games tape packed with ten games of my own design. Send $20 for the bargain tape (includes p&p) to: R. Lyon, Jamieson post office 3723. Ph. (057) 770-554.

LASERLINK
LASERLINK
LASERLINK

INTRODUCES 24 "NEW" COMMANDS FOR YOUR VZ200/300
Your VZ can now be Level II compatible for only **$35**
Telephone Gavin Williamson – (049)62 1678 – for name of agent in your state
LARGE RANGE OF SOFTWARE & HARDWARE AVAILABLE FOR VZ

FOR SALE VZ 200/300 users short basic program. Save Binary Programs to tape and disc. Send $5 to P. Brennan, P.O. Box 334, Mordialloc, Vic 3195.

ETI Jun. 88. p21.

APC November 1987 Page 297

HARDWARE REVIEWS

Apr.	83	YCU	56-59	Texet TX-8000. (Bennett)	(3)
Apr.	83	APC	58-66	VZ-200. (Hartnell)	(5)
Apr.	83	CC	38-43	Review of VZ-200.	(3)
May	83	CC	26-30	Video Technology VZ-200 PC. (Ahl)	(3)
Jun.	83	EA	137	New low-cost computer - VZ-200.	(1)
Jun.	83	ETI	30	Dick Smith colour computer.	(1)
Jun.	83	YC	6	DSE VZ-200.	(-)
Aug.	84	PCG	12	VZ-200.	(-)
Jul.	83	ETI	32-7	DSE's personal colour computer. (Harrison)	(3)
Jul.	83	EA	130-3	The VZ-200: colour, graphics and sound. (Vernon)	(4)
Jul.	83	PCN	16	Timing the Laser's phazer. (Stokes)	(1)
Sep.	83	WM	40	Laser.	(-)
Sep.	83	BB	18-20	Dick Smith VZ200: good value. (Fullerton)	(3)
Aug.	83	YC	20-33	Cash and Carry Computers. (Bell)	(9)
Sep.	83	CC	202-4	Review of VZ-200 and PP40.	(1)
Oct.	83	APC	77-8	VZ-200.	(1)
Oct.	83	WM	135	Texet TX8000.	(1)
Oct.	83	CT	12	The Laser 200.	(-)
Dec.	83	CT	11	Laser 200.	(-)
Nov.	83	CT	37-40	A look at the Laser. (Green)	(4)
Nov.	83	WM	42-108	The Laser - a shot in the dark.	(3)
Nov/Dec	83	SYN	17-22	VZ-200. (Ahl)	(2)
Feb.	84	CC	218-21	Laser PP40 Printer/Plotter.	(2)
Spring	84	MC	52-4	Laser 200. (Green)	(3)
Jun.	84	EA	12-9	Buying your first computer. (Vernon)	(6)
Aug.	84	EA	30-3	An important role for small computers. (Williams)	(4)
Oct.	84	PCG	82-87	Home micro supertest. Pt. 3 (Bollington)	(5)
Nov.	84	PCG	14-19	Home micro supertest. Pt. 4 (Bollington)	(4)
Nov.	84	EA	78-80	VZ-200 as a WP (DSE E&F tape WP). (Williams)	(2)
Dec.	84	CHC	28-31	Review of video games consoles.	(4)
Mar.	85	EA	31-33	Back to the VZ-200. (Williams)	(1)
Jul.	85	ETI	102-6	Dick Smith's new VZ-300. (Rowe)	(5)
Aug.	85	EA	22-7	WP on the new VZ-300. (Williams)	(5)
Dec/Jan	86	PCG	11-15	How to buy a micro - VZ-300 compared.	(4)
Aug.	86	AHC	38-39	Computers for the Rest of Us. (Roberts)	(2)
Nov.	86	AHC	44	Letter. (Kennedy)	(-)
Dec.	87	YC	20-21	VZ-300. (Hartnell)	(2)
Dec.	87	YC	78	VZ-300	(1)

£98 TEXET TX-8000

TEN YEARS AFTER Texet drove Sinclair out of the pocket calculator market the TX-8000 is ready to take on the ZX-81 and the Spectrum. As with the calculators Texet hopes to win customers by aggressive pricing. But although the £98 TX-8000 is now the cheapest colour micro — by a whisker from the Oric and by £27 from the Spectrum — it has only 4K RAM as opposed to the 16K of its rivals.

The Z-80 based TX-8000 has a specification that, on paper, looks very good compared with the ZX-81. When it is compared with, for example, that of the Oric, then a number of weaknesses become apparent.

Of the three colour computers under £125 — the Spectrum, Oric and TX-8000 — the TX-8000 is the largest. It case is made of a cream plastic, which feels more brittle than the plastic used for its rivals — but it would still require an act of malice to break it. The design of the case is not as polished as that of its rivals, but it does have a gently sloping front which means the keys actually face the user.

The dimensions of the case are 12in. wide by 6in. deep, 2in. high at the rear and 1in. high at the front. The panel containing the keys is dark brown and sunken into the body. There are 45 keys in a rubber keyboard which is very similar to that of the Spectrum. Not only do the keys squash down in the same way they even have that distinctive clammy feel to them. If anything the Texet keyboard feels worse than the Spectrum's.

Individual keys are smaller than on the Spectrum, but there are more of them. Keyboard layout is based on the usual QWERTY typewriter formation, which the TX-8000 mimics better than the Spectrum. This necessitates fewer key depressions, especially in the case of punctuation symbols which can only be achieved by a shifted key on the Spectrum but have their usual typewriter keys on the TX-8000.

Above the first eight number keys there are the corresponding colour names; yellow, blue, red, buff, cyan, magenta, orange and green. This is the same colour set as on the Spectrum but with the addition of buff and orange. Interestingly, there is no black or white, which look in theory to be unobtainable.

When using the keyboard the letter pressed is what appears on the screen, even though certain Basic keywords are printed above and below the keys. The keywords are accessed by the kind of finger gymnastics that put me off the Spectrum when it first appeared. Alongside the keyboard is a power light which tells you when the machine is on, which sometimes is not apparent from looking at the screen.

On the right-hand side of the machine is a rocker-type switch, to turn the power on and off. This is a welcome feature, as anyone using a Spectrum or ZX-81 will know that the continual insertion and removal of the power supply plug eventually works it loose. So a cold reset — that is a reset of the computer which clears the RAM — is a simple operation.

Although the machine is marketed in this country as the Texet TX-8000, elsewhere it is known as the Video Technology VZ-200. This is taking badge engineering to new heights. The Texet is exactly the same as the Video Technology machine except for the VZ-200 badge. Both machines are manufactured in Hong Kong, the factory-door price of the VZ-200 being $66 — less than £45.

The real significance of this similarity is that there are a number of interesting peripherals available for the VZ-200, which will work with the Texet. These include: 16K and 64K Ram extensions, joysticks, printer, light-pen, Modem, disc-drives and bar-code readers. There is also an interface unit which allows you to use any standard text or graphics printer. All these add-ons are manufactured by Video Technology in Hong Kong and will be available in the U.K. from Texet. Projected prices are: printer, £129; 64K Ram expansion, £52; £8 for a single paddle and £60 for a pair of cordless remote control joysticks.

Opening up the inside of the Texet is like digging in the garden of the Cricklewood house of horrors. A number of vaguely familiar objects are recognisable amongst the mess even though all the identifying codes on the chips have been painted out to preserve their anonymity. There is a black and white model of the VZ-200 in Hong Kong and one look inside the case of the Texet shows that it is basically a black and white computer that has been converted for colour. The colour circuitry is antique by the standards of the Oric or the Spectrum, with a large number of presets, pots, coils and resistors.

On the rear of the machine are the usual power and TV output sockets. In addition there is a tape socket, which unlike conventional tape sockets is a stereo jack socket — the kind used on portable hi-fis. This connects to two mono jack plugs, red and black, the red one being the Ear connection and black the Mic.

There is also a monitor output — which will not work with most monitors. Also along the back of the machine, but covered by a couple of aluminium panels are the bus expanders. One is marked Memory Expansion and the other, Peripheral. This may imply that only one peripheral can be connected at a time. The panels are attached to the computer by two tiny screws.

Power for the micro comes from a transformer which would plug straight into the power socket except that it has a two-pin electric shaver-type plug. This needs a special adapter to enable it to be used with domestic U.K. power sockets. Unfortunately the pennies this adds to the price of the micro makes the Texet only a pound cheaper than the Oric.

Because the transformer itself is attached to the plug its weight causes it to work its way out of the socket. While this is not likely to be dangerous, due to the insulation on the pins, it does mean that a programming session can be ruined and all work lost due the the resulting power failure.

When the machine is powered up the message:

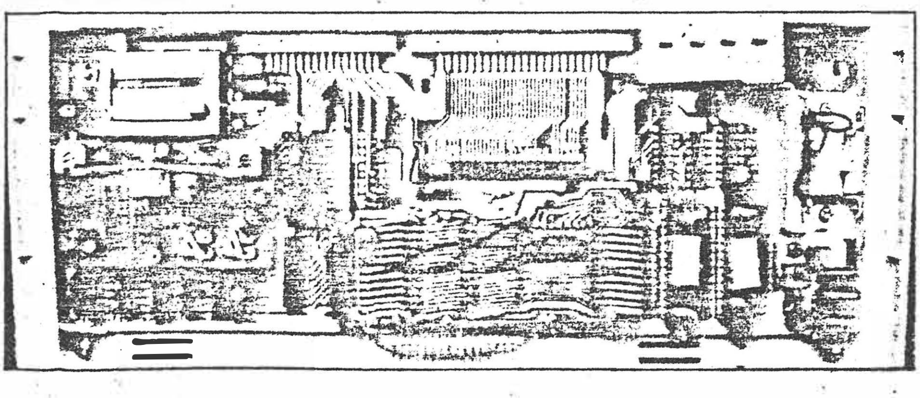
```
VIDEO TECHNOLOGY
BASIC V1.0
```

appears. The letters are in light green on a darker green background with the whole surrounded by a black border. The cursor — a square of light green, flashes on and off. If the on/off switch is flicked momentarily to the off position and back again a bizarre effect on screen is caused by the memory-mapped screen area of RAM being filled with garbage.

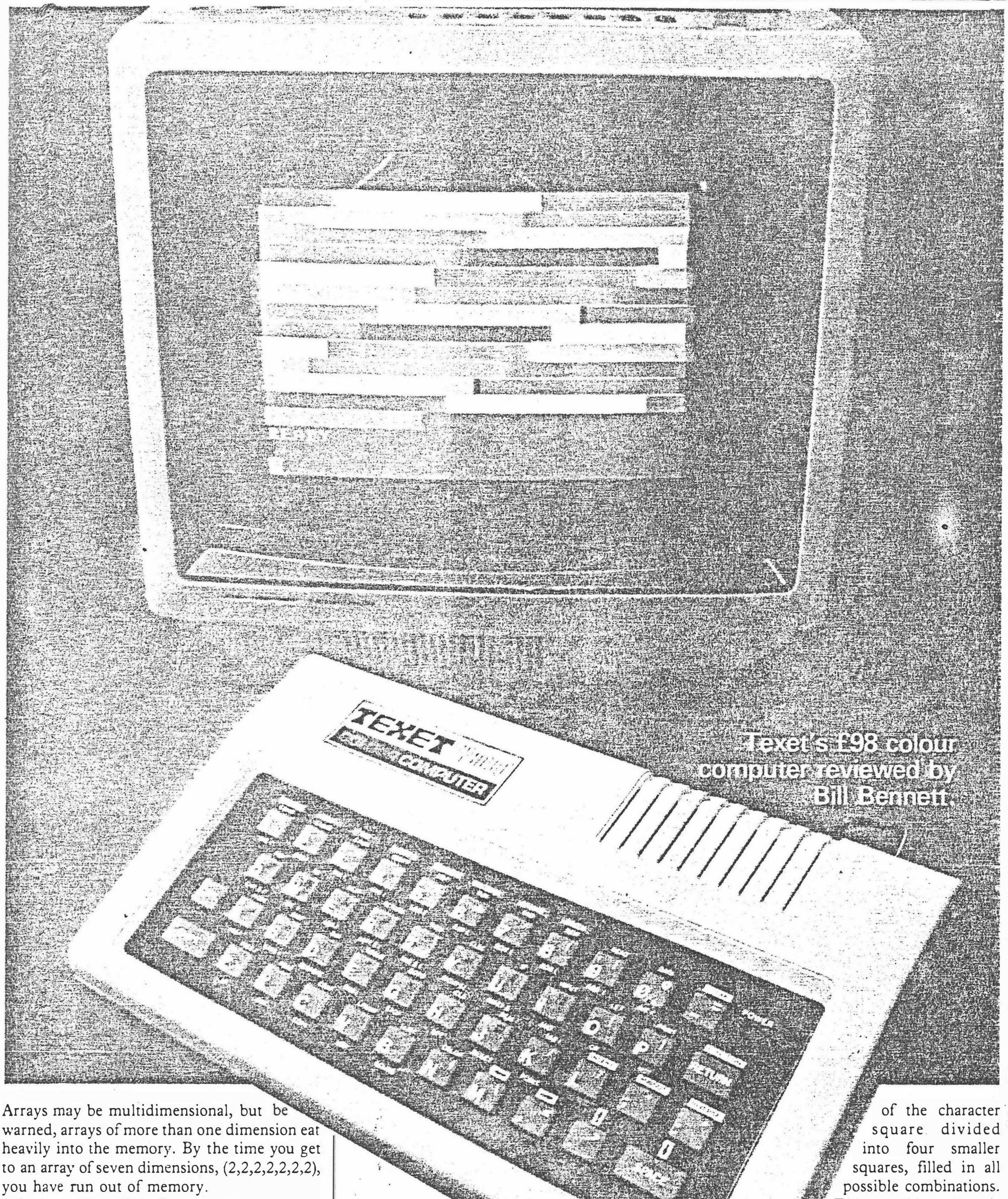
The TX-8000 has only 4K RAM — and 2K is available for programs, the other 2K is for the screen. The maximum size of a numeric array defined in a Basic DIM statement is 1313 locations and a string array can hold 1751 strings.

Of course should you decide to use arrays that big, there will not be any room left for the program. Anyone who has used the ZX-81 might think 2K is a lot of memory, especially when it does not have to store the display as well. If you were a bit tight for space, you could try storing numbers as strings though.

With internal circuitry that looks like this (below) it is not surprising that colours are displaced by half a character on screen.



 (UK.) 1 of 3



Texet's £98 colour computer reviewed by Bill Bennett.

Arrays may be multidimensional, but be warned, arrays of more than one dimension eat heavily into the memory. By the time you get to an array of seven dimensions, (2,2,2,2,2,2,2), you have run out of memory.

A simple line of Basic, such as:

```
10 x=20
```

only takes up four bytes so a reasonable program can be squeezed into the memory. However this compares very unfavourably with the Oric, which is only a fraction more expensive, but has a nominal 16K of RAM.

The organisation of the video memory is interesting, in the normal text mode — which is called from Basic by the command Mode(0), the first 512 bytes of video memory store the ASCII codes of the screen characters.

Any of the machine's character set of 255 characters can be Poked into this memory, and of course it is the area that the Print command uses.

All the usual characters appear in the set, together with their inverses. In addition there is a subset of graphic characters which consist of the character square divided into four smaller squares, filled in all possible combinations. This graphic subset is repeated four times. When the machine is initially turned on this character set appears in four different colours but use of the Color command — which is similar to Ink on the Spectrum — changes this, and the four sets seem to change to arbitary colours.

(continued on page 59)

2 of 3. (UK.)

(continued from page 57)

Colour only affects the graphic symbols. There is no provision for printing words or letters in colour. What is strange is a lack of black or white on the screen. In practice the colour designated as Buff is slightly off-white and for most purposes can be used in its place. Black can be obtained only as the other colour in the graphic symbol character set.

There is a major problem with the colour on the Texet, it seems that each of the colours is attributed to a character space that is displaced half a character to the right of the printed character. That is, the printed characters and their assigned colours do not match up on the screen. This could be a fault in the review machine, but looking at the colour circuitry within it is not surprising.

It is a shame about the colour location problem, because the colours themselves are the brightest on any of the cheaper colour computers. The red is a little darker than it should be, but the blue and orange are as luminous as Day-Glo colours. The colours can be changed by altering the controls of your TV set, but the alignment problem cannot be ironed out.

The graphics characters can be printed or Poked on to the screen by using their character codes, but they are also accessible from the keyboard. To print them in, say, a pair of quotes inside a Print statement, you have to press both shift and control at once, then the relevant graphics key. Graphics are printed on the key switches, so you have some idea which one you are using. On the Z key a graphic block is printed which does not correspond to the character printed by that key, and keys: x, c, v, b, are merely repeats of characters that can be found elsewhere and consequently are not marked.

When printed directly from the keyboard the graphics characters appear in the default light and dark green colour set. After a Color command however they will be printed on the screen in that colour. The characters print on to the screen extremely fast in this mode, a thousand colour graphic strings taking less than 20 seconds. But there is a price to pay. A string can only contain graphic characters of one colour, and that colour is always the colour specified by the preceding Color command.

In the text graphics mode, mode 0, the screen is organised into 16 lines of 32 characters. This compares with 24 lines of 32 on the Spectrum — or more correctly 22 usable lines, and 28 by 40 on the Oric — which is a Prestel-like display.

High-Resolution mode, mode 1, is not really high-resolution at all. There are only 128 by 64 pixel locations, which is not much better than some — albeit much more expensive — microcomputers' text mode. This takes up the entire 2K of the video memory, which is interesting because 128 × 64 is not 2K, but 8K.

It works in a way that is similar to the text mode. There are 32 columns and 64 rows, each of which can have any value up to the eight-bit limit of 255. In text mode these normally represent characters, but in mode 1 they represent short graphic strings of four pixels, arranged in a line one after the other. Poking a value into one of these locations specifies the colour of each of those four pixels.

Obviously not all possible combinations of the eight colours in four pixels can be accommodated — there are 4,000. Unfortunately thanks to the colour misalignment, colour is not always visible in this mode.

Light green is the only possible background allowed in the so-called high-resolution mode. To let you know that the mode has changed from low-resolution/text to the pseudo high-resolution the border colour changes from soot black to the same lime green as the rest of the screen. This is to avoid any confusion between what might be called low-resolution 1 and low-resolution 2.

So bad is the colour misalignment that when a sine curve is displayed on the screen, it appears as black on the lime green background, with a hint of whatever the chosen colour was around the edges. This makes a mockery of the TX-8000's ability to display any of its eight colours at any one of the 128 by 64 locations.

Poking to the display is a complicated business in this mode, so there are adequate Basic commands to handle the graphics. They are Set and Reset — which plot and unplot points on the screen, and Point which examines a position and tells you if it is on or off.

Despite the ventilation both in the top of and under the case, the machine can become very hot. This could be due to the poor thermal contact of the heat sink, which was only loosely connected to the power supply semiconductor. This can cause problems. When the machine was turned off momentarily — due to the transformer falling out of the socket — the television had to be retuned to obtain a picture.

TX-8000 Basic is a fairly standard version of Microsoft Basic. It holds few surprises but does have some refinements that, if omitted, would make the Texet a very old-fashioned machine indeed. There is the Step to go with For . . Next, and the Else to supplement the If . . Then. As far as structures go, the TX-8000 is a non-starter.

Cassettes are loaded with the CLoad command, which causes the machine to print Bad on the screen whenever a load fails. Loading is extremely difficult because unlike the Spectrum there is no screen display to let you know how well the load is going.

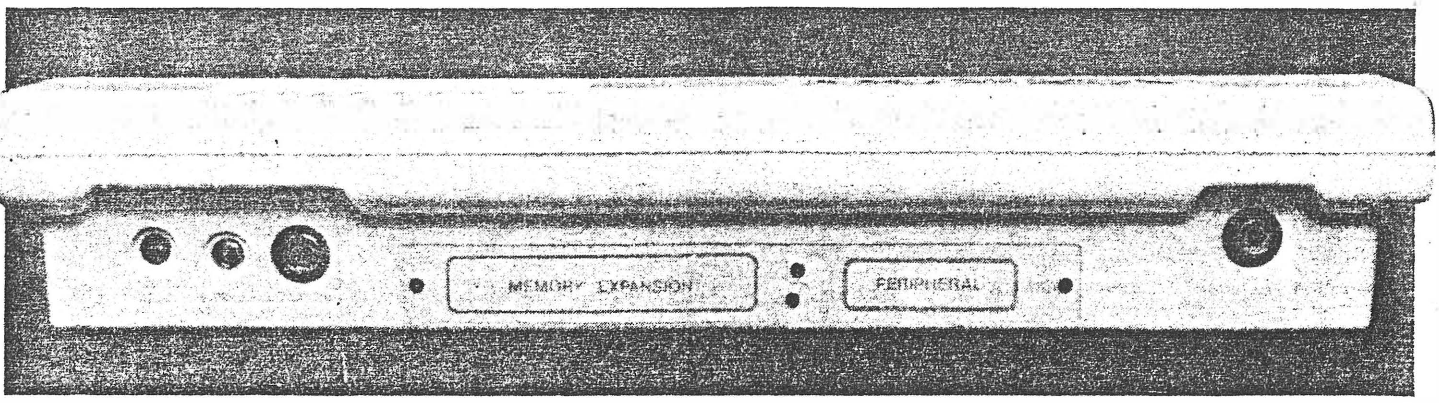
CSave is accompanied by a Verify command, which no self-respecting micro would be seen without these days. All the tape operations are performed at 600 baud which is faster than the ZX-81 but slower than the Spectrum — the Oric allows you to choose speeds. The speed could be at the root of the loading problems but more likely the main offender is the power socket, which is located right next to the cassette socket.

Basic programming lines cannot be longer than two screen lines. If you try entering one longer you simply lose it without warning. The Sound command is feeble compared to the Oric. All it can do is play rather quiet tones — there is no loudspeaker. The Sound command has two parameters, the first being the pitch. This can have any integer value between 1 and 31. If a decimal number is input it simply truncates and plays the next one down. The second parameter is the length of the tone and this is variable between one and nine.

Numbers can only be printed to six significant figures which means that should a business be in such bad shape that it decides to install a TX-8000 as a computer, it will never be able process debts greater than £9,999.99. To ensure neatness trailing zeros are suppressed. ■

CONCLUSIONS

- The Texet TX-8000 may enjoy a brief period of fame as the cheapest colour computer around but too many compromises have been made.
- The colour display on the screen needs tidying up as does the internal construction of the Texet. If this was done then the peripherals available for the TX-8000 — especially 64K expansion for £52 might make it worth a second glance.
- The shortcomings of the £98 Texet make the high standards of the £99 Oric and the £125 Spectrum seem all the more remarkable.



3 of 3. (UK)

BENCHTEST
PERSONAL COMPUTER

VZ 200

Dick Smith has surprised Australia with a price/performance breakthrough in home computers. Tim Hartnell reports.

INTRODUCTION

A colour computer for less than $200? It sounds hard to believe, but Dick Smith has done it with the VZ-200, which will be released in Australia towards the end of May. Manufactured in Hong Kong by Video Technology Ltd to Dick Smith's specifications, this small computer is certain to send shivers of dismay up the spines of dealers in other small computers, such as the VIC-20 and the Sinclair Spectrum.

HARDWARE

The VZ-200 is tiny. Smaller than a telephone directory (29cm long, 16.5cm from front to back, with a height of just 2.5cm at the front of the keyboard, rising to 5cm at the back), the unit is built from cream plastic. The computer is light, but does not feel excessively fragile.

The keys are rubber (much like the Spectrum keys), in light brown, with easy-to-read white legends on them. A red LED in the top right hand corner of the keyboard lets you know the machine is on (and the on/off switch is located under the 'lip' of the keyboard, down the right hand side, in a position where it would be almost impossible to turn it off accidentally).

Each key has one or two things written on it, generally a letter (the computer works all in upper case on the screen) and a symbol (such as & or *), or a graphics element. These are a series of squares, each the size of a letter, with various quarters blocked off, to give a total of 15 different fairly coarse shapes. Above most keys are key words (such as FOR, INPUT and PRINT) while below the keys is another set of words, the functions (such as CHR$, SIN and LOG).

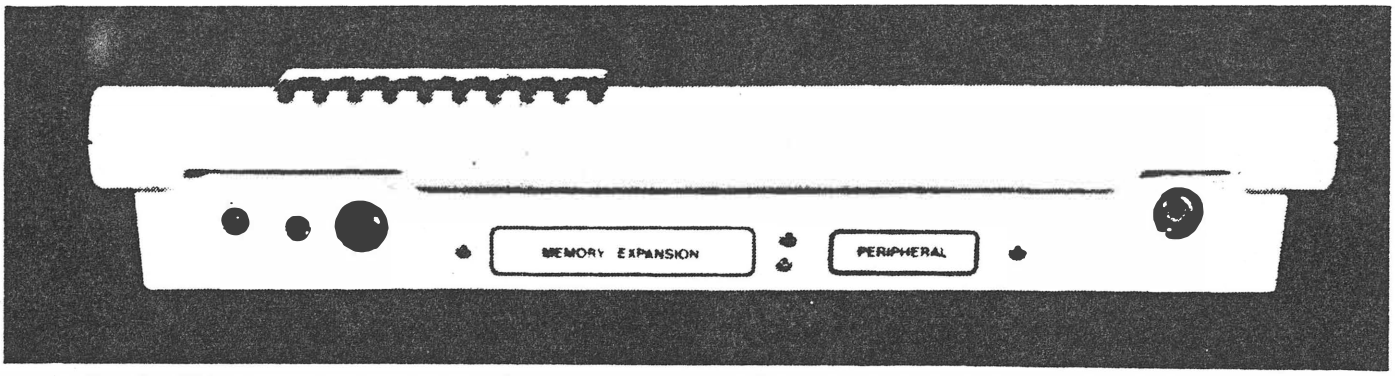
This single element on the VZ-200 shows the influence of Sinclair, who pioneered the 'single touch, key word' entry system back with the ZX80. In contrast to the ZX81 and the Spectrum, the VZ-200 does not demand you use the single-touch keys. If you feel happier typing out words in full (which is almost certain to be the case if you decide to move from another computer to the VZ-200), this Dick Smith machine will allow you to do so. You can even mix single-touch entered words, and spelt out words, in the same program line.

As you can see from the photograph of the keyboard, there is a SHIFT key in the bottom left hand corner, and above that is the control key (marked CTRL). If you hold down CTRL and then touch another key, you'll get the key word written above the key. Underneath the power LED is the RETURN key, and written above this is FUNCTION. If you hold down the CTRL key, then press RETURN/FUNCTION, and then press a key, the word underneath the key will appear on the screen.

The keys numbered one to eight have a further set of words above them. These are the colours (green, yellow, blue, red, buff, cyan, magenta and orange) and above these is the message 'Mode 0 only'. We'll be discussing the modes in the software section.

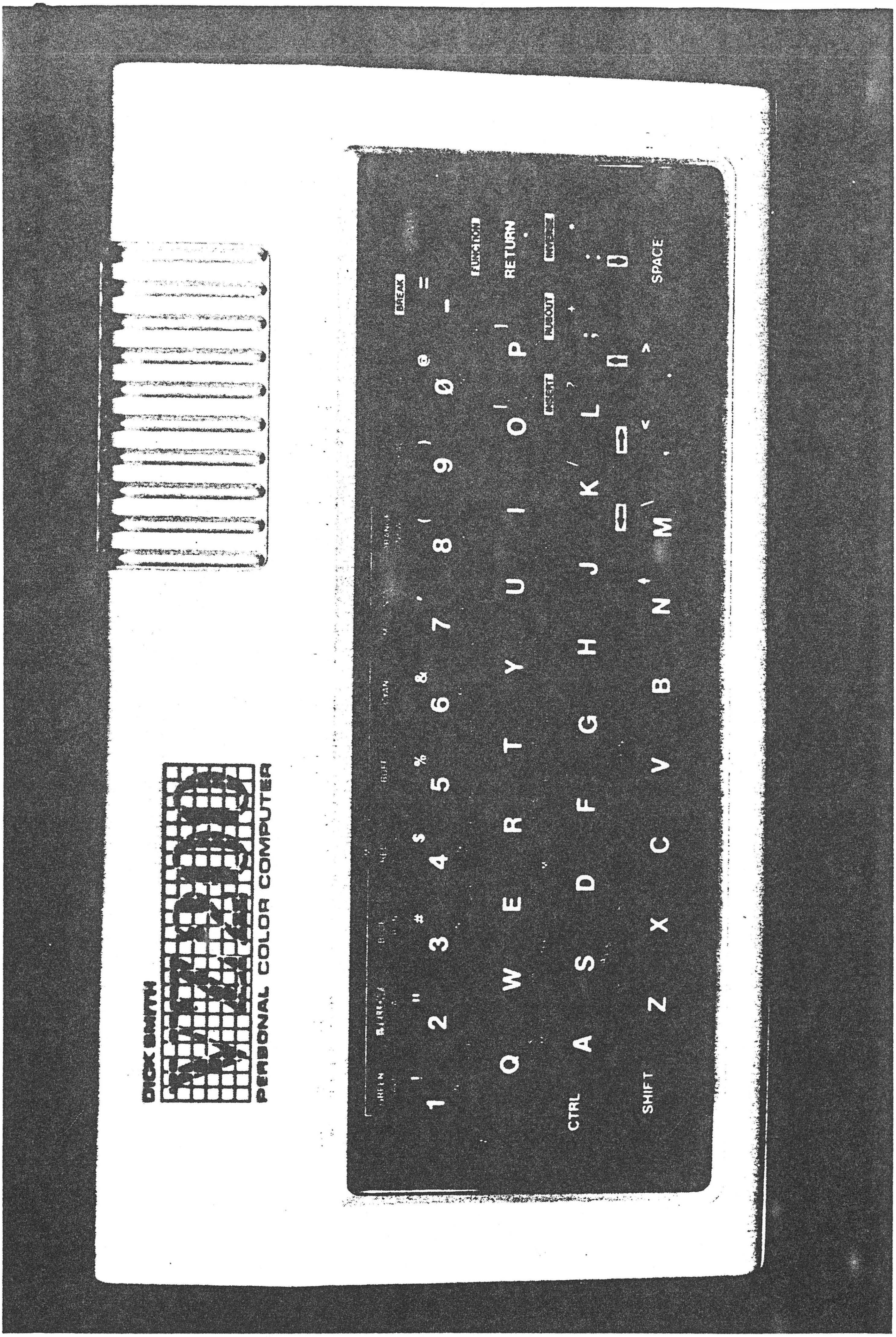
You may feel, on reading this description and looking at the keyboard and its bewildering array of words and symbols, that the VZ-200 will be extremely difficult to get used to. I felt that way when I first tackled the Sinclair Spectrum keyboard (which is even more complicated), but discovered that it became remarkably easy to use after a very short time. I am sure the same thing will happen with the VZ-200. Even if you start programming on it without using the one-touch key word entry system, you'll probably soon find yourself using some of the 'pre-programmed' words (such as RUN above the 6 key, and LIST above the 5) rather than type out the whole word every time. From there, it won't be long before you're introducing more of the single keys into your programming.

The keys feel good. Although they are a sort of 'dead rubber', they are extremely responsive, requiring only the slightest touch to trigger (in contrast to the Spectrum, whose keys have to be squeezed slightly to get the finger pressure to register). The keyboard beeps when each key is pressed, giving good audio feedback to your typing, although there is no tactile feedback at all. Of course, a keyboard of this type can never really compete with a real keyboard such as the one provided on the VIC-20, but when you're buying a colour computer for $200, you have to



Left: The VZ-200 in actual size less about 10%. Above: The rear end showing sockets for the monitor, TV, cassette and plate covered edge connectors for peripherals and additional memory.

Apr 83 4(4) 58-66 1 of 5.


DICK SMITH
PERSONAL COLOR COMPUTER
BREAK
RETURN
CTRL
SHIFT
SPACE

 Apr 83 2 of 5.

be willing to make some compromises.

The computer comes with a separate power unit (producing 10 volts at 800 milliamps) which plugs into the rear of the machine. This is supplied with a generous three metre cable (unlike some computers which come with leads so short manufacturers must imagine you like sitting on your power point to do your computing). A much shorter (around a metre) cable is provided to connect a cassette player to the VZ-200. A 'stereo' plug goes into the computer socket which is marked TAPE and the other end of the cable branches into two 3.5mm plugs, one each for the earphone and microphone sockets.

There are two video outlets. One connects your computer to a standard television, and while I did have a little difficulty locating the correct channel for the picture, once I'd found it, the picture was clear and steady, and did not drift. The second video output is to drive a monitor, allowing a somewhat superior picture to be produced. Providing both these outlets is a good touch, allowing you to upgrade your picture quality if you have a monitor, without having to adapt the modulator output for it.

When you turn the computer on, the screen comes up with a black border framing a green central area, with white writing (VIDEO TECHNOLOGY BASIC V1.1 READY). The letters tend to be fairly large and square, rather like those produced by the TRS-80 Color Computer. The cursor is a flashing white oblong.

The computer comes with 8k of RAM on board of which approximately 6k is available to use (in contrast with the VIC, which has only 3.5k or so of user RAM on the unexpanded model).

There are two sockets at the back of the machine which are protected by small panels, held in place by a couple of Philips screws. They are marked 'memory expansion' and 'peripherals'. The 16k memory unit (which will cost $79.00) is rectangular, somewhat larger than a cigarette box, in the same pale cream plastic as the computer. The memory module fitted easily into place, and sat in position fairly firmly, although I would not advise waving the computer around in the air with the extra memory in place.

The 'peripherals' bus will take plug-in ROM cartridges. As well, it can be used to interface (via an optional unit which will sell for $49.50) to any Centronics-type printer.

The computer case is held together with six screws, fitted underneath. There are a few ventilation grills in the base of the machine, which is supported a few millimetres above the table surface with four tiny rubber feet. Inside the computer, much as you'd expect, there is the normal assortment of chips and other components which are always incomprehensible to people like me who find the whole hardware area a forbidding jungle.

The keyboard unit, which is fastened solidly to the top half of the computer case, is linked with the main body of the machine via a short, 16-wire cable. It appears it would be a simple job to tap into this to connect up a larger, full key keyboard if you wanted to do so. There is a small heatsink which lies under the grill you can see in the left hand corner of the computer, when looking at it from the front. I am constantly surprised by how tiny modern computers are, and the VZ-200 reinforces that surprise. The case isn't even full.

The memory map is as expected. The Basic ROM occupies the first 16k (up to 16384, 3FFF) with the next 14k or so divided up into 10k for the ROM cartridges, 4k for the keyboard, cassette port, video controller and sound, and 2k video RAM. Next comes the inbuilt user 6k RAM. The memory of the unexpanded machine ends at 36863 (8FFF). The computer can be expanded by a further 16k, using the module mentioned earlier, to 65535 (FFFF).

SOFTWARE

The computer has a 16k ROM, of which 8k is a good implementation of standard Microsoft Basic, with the second 8k holding the commands for accessing the sound and colour. Additional text and graphics commands, such as PRINT @ (to position a character in an exact

position on the screen; an ideal and easy way to create moving graphics) and PRINT USING are also supported.

As I said earlier, the screen comes up green, with white writing. Holding down the CTRL key, then pressing the key second from the bottom right hand corner (marked INVERSE) produces green letters on little white oblongs. These inverse letters come out as lower case letters when the computer output is dumped to a printer. Holding down CTRL, then pressing INVERSE again changes the letters back to white on green.

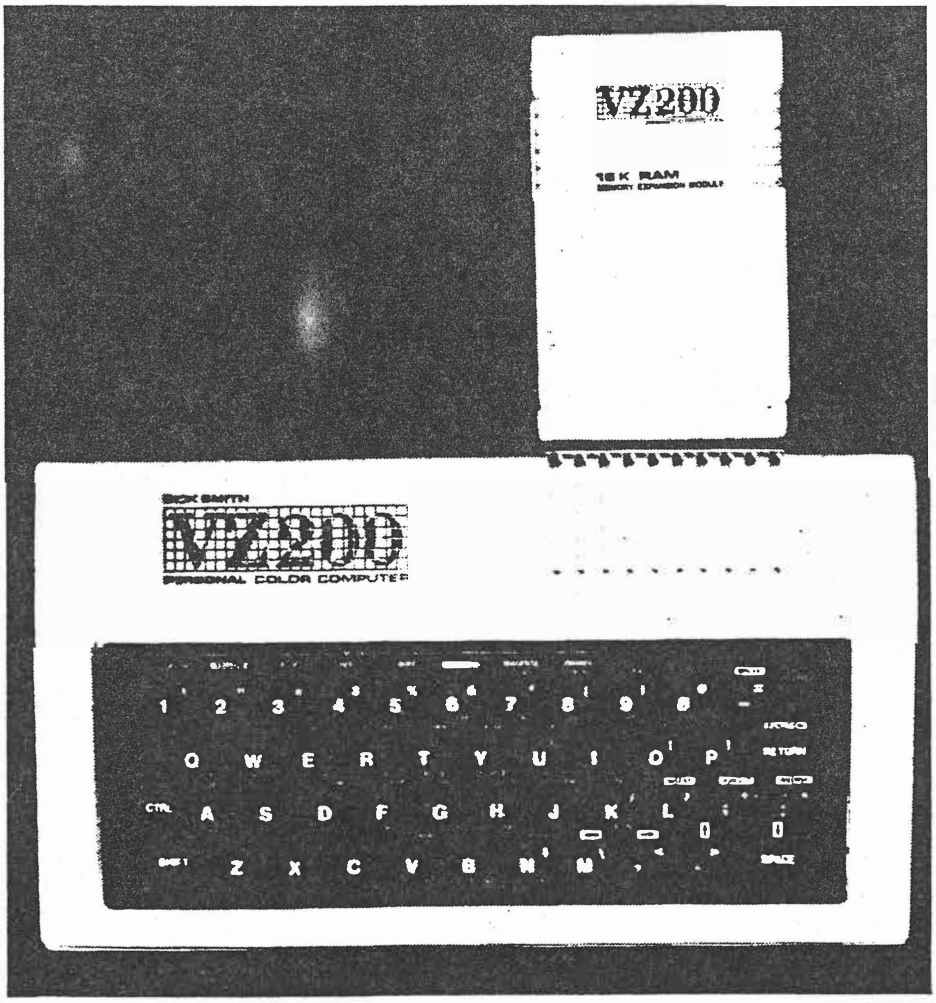
The VZ-200 works in two graphics modes. The display in text mode is 32 by 16, while in the higher graphics mode you have a resolution of 128 by 64. This is not particularly high, but is adequate for many applications.

The computer defaults to the text mode (MODE 0) when you first turn it on. The colours are easy to use in this mode. You simply include the command COLOR n,m (where n is a number between one and eight, and m is either zero or one) and the VZ-200 prints the following text in that colour.

There are only two background colours, and these are controlled by m. The two backgrounds are green (0) and orange (1). COLOR 1 will switch the background colour, no matter which one is currently in place. The computer will stay in the specified colour until a new one is evoked.

The cursor position is controlled by four arrowed keys (all grouped together conveniently in the bottom right hand corner of the screen). Holding down CTRL, then pressing one of these will cause the cursor to move rapidly about the screen, inverting any letter or symbol it moves over. Once you've got the cursor where you want it to be to edit a program line, you can either use the INSERT key (still holding down CTRL) to make room for new material you wish to add (the new spaces stream off from the right of the cursor) or RUBOUT (which 'draws in' material from the right of the cursor, causing it to vanish underneath the cursor). The arrow keys are easy and swift to use, and allow program lines to be edited simply.

The SET and RESET commands are used in the higher resolution mode to turn on (SET) and off (RESET) specific points on the screen. The command is of the form SET (X, Y) where X is from zero to 127, and Y is zero to 63. The dots are printed in specific colours. (The Spectrum, by contrast, boasts a 256 by 172 screen, but the colour resolution is only 32 x 22). POINT is used in conjunction with SET and RESET to return the state of a particular position (that is, to tell if it is 'turned on' or not).



The 16k RAM expansion module is quite large as compared with the VZ-200 itself.

Of course, PEEK and POKE can be used to directly access the display file, for fast moving graphics. (The display file starts at 28672 in both modes, ending at 29183 in mode 0 and 30719 in mode 1). You need to POKE with numbers between 127 and 255 to get coloured graphics, while POKE codes 64 to 127 hold the inverses of the letters, numbers and symbols which precede 64.

SOUND

The musical output of the computer, and the beeps when you press the keys, come from a tiny inbuilt sound device. The volume is just adequate (although louder than the Spectrum's sound) but is far better than having no sound at all. The VZ-200 sound is, however, woefully inferior to the sound produced through the TV loudspeaker by the VIC-20, where you have three voices and white noises to play with (even if the VIC sound must be accessed through tiresome and complex POKE statements).

The VZ-200 sound is controlled by a SOUND statement, of the form SOUND n,m – where n is the pitch (1 to 31) and m is the duration (1 – shortest – to 9). The following, two-line program will put the VZ-200 through its musical paces forever:

```
10 SOUND RND(31), RND(9)
20 GOTO 10
```

CASSETTE HANDLING

Cassette handling on the VZ-200 is quite sophisticated. The computer dumps the programs to cassette with the command CSAVE "nnnn", where "nnnn" is a file name. The command CLOAD – again qualified by a file name – is used to get programs back from tape into the computer. The computer will print up the names of other programs found on the tape before the one you have specified, and while loading prints up the message LOADING:nnnn. I have used (and cursed at) a variety of cassette interfaces in my years of working with computers. The VZ-200 performed faultlessly for me once I had worked out the right setting for my cassette recorder, and when I used good quality audio or computer cassettes. It did not work so well with ordinary, cheap audio tapes. Tapes made by companies like TDK should give consistently good results.

A third cassette command, VERIFY, is provided so that you can check the quality of a SAVE before wiping the program from the computer. This compares the program on the tape with the one in the computer and reports VERIFY OK if the two correspond exactly.

Many Basics support the CHAIN

Apr 83 4 of 5.

command (used as CHAIN "nnnn") which is a 'load and go' command. The command finds the specified program on the tape or disk, loads it, and then starts running the program automatically. The VZ-200 command CRUN provides this facility.

The hash (#) symbol, in conjunction with INPUT and PRINT, can be used to put and get file data from tape. This is an advanced feature which could substantially extend the potential uses of the VZ-200.

DOCUMENTATION

The computer comes with a hefty manual, which covers the entire VZ-200 Basic language, touching briefly (but relatively clearly, given the complexity of the subjects) on PEEK and POKE, INP and OUT (for returning the content of a port, and for sending values to an I/O port) and USR (to call a machine language subroutine).

The manual starts with a two-page explanation of the major parts which make up a computer system. This is not needed in order to use the computer, and first-time users are advised to skip over it (as it contributes nothing to getting your VZ-200 up and running) with the idea of perhaps coming back to it later.

The manual is clear. It has been written by Video Technology under strict instructions from Jime Rowe of Dick Smith Electronics. The intention has been (and this is supported by the notes I saw which have gone back and forth from Hong Kong to Australia) to make everything as clear as possible for the first-time user.

A book 'Getting Acquainted With Your VZ-200', is in preparation. This will introduce programming in a more informal style than that provided by the manual, which will remain the standard source of information for users.

A series of software packs, mostly games, will shortly be available from the manufacturer, and Dick Smith has commissioned several more original programs from Australian programmers. A users' club has been organised (with the co-operation of, but not under the control of, Dick Smith) and members will be entitled to free copies of the club's newsletter.

CONCLUSIONS

Overall, this is a great little machine, and one that is likely to change the face of Australian personal computing. With one move, it has attacked the market of every machine under $1000. Assuming the promised support materialises (and Dick Smith has a reputation for delivering) VZ-200 users should shortly find that their computer is better supported (in terms of available software, books, magazine articles and a users' club) than any other machine in this country.

Purchasers who buy the machine, knowing that for $200 they won't be getting the sound output or keyboard quality of a more expensive machine, will probably be well-pleased with their purchase.

When the editor of *APC* came over to my place to see the machine while I was writing this review, he said: "I'm certainly going to buy one." I am sure this will be the reaction of a great number of Australians. I have a feeling we are going to be hearing a whole lot more of the Dick Smith VZ-200 Personal Color Computer in the coming months.

BENCHMARKS

The standard eight Benchmark tests were applied, and produced the following results:

BM1 loop 1.5 seconds
BM2 loop/addition 6.7 seconds
BM3 loop/addition/arithmetic 17 seconds
BM4 loop/addition/arithmetic numbers 17.5 seconds
BM5 as above/subroutine call 19 seconds
BM6 as above/dim/inner loop 31 seconds
BM7 as above, fill array 47 seconds
BM8 trig functions 72 seconds (1000 loops).

Average – 26.5 seconds.

Comparing these with the VIC-20, we find that they are very close, with the VIC's average time of 28.7. However, they are significantly faster than the Spectrum, coming in with an average of 58.5 for the eight Benchmarks. As Dick Pountain pointed out in *APC* in November, 1982, the result of the Benchmarks tests does not necessarily prove very much, although the results are interesting.

TABLE OF RESERVED WORDS – VZ-200

ABS AND ASC ATN
CHR$ CLOAD CLS COLOR CONT COPY COS CRUN CSAVE
DATA DIM
ELSE END EXP
FOR
GOSUB GOTO
IF INKEY$ INP INPUT INT
LEFT$ LEN LET LIST LOG LLIST LPRINT
MODE MID$
NEW NEXT NOT
OR OUT
PEEK POKE POINT PRINT
READ RED RESET RESTORE RETURN RND RUN
SET SGN SOUND SIN SQR STEP STOP STR$
TAB TAN TO THEN
USING USR

VZ-200 TECHNICAL SPECIFICATIONS

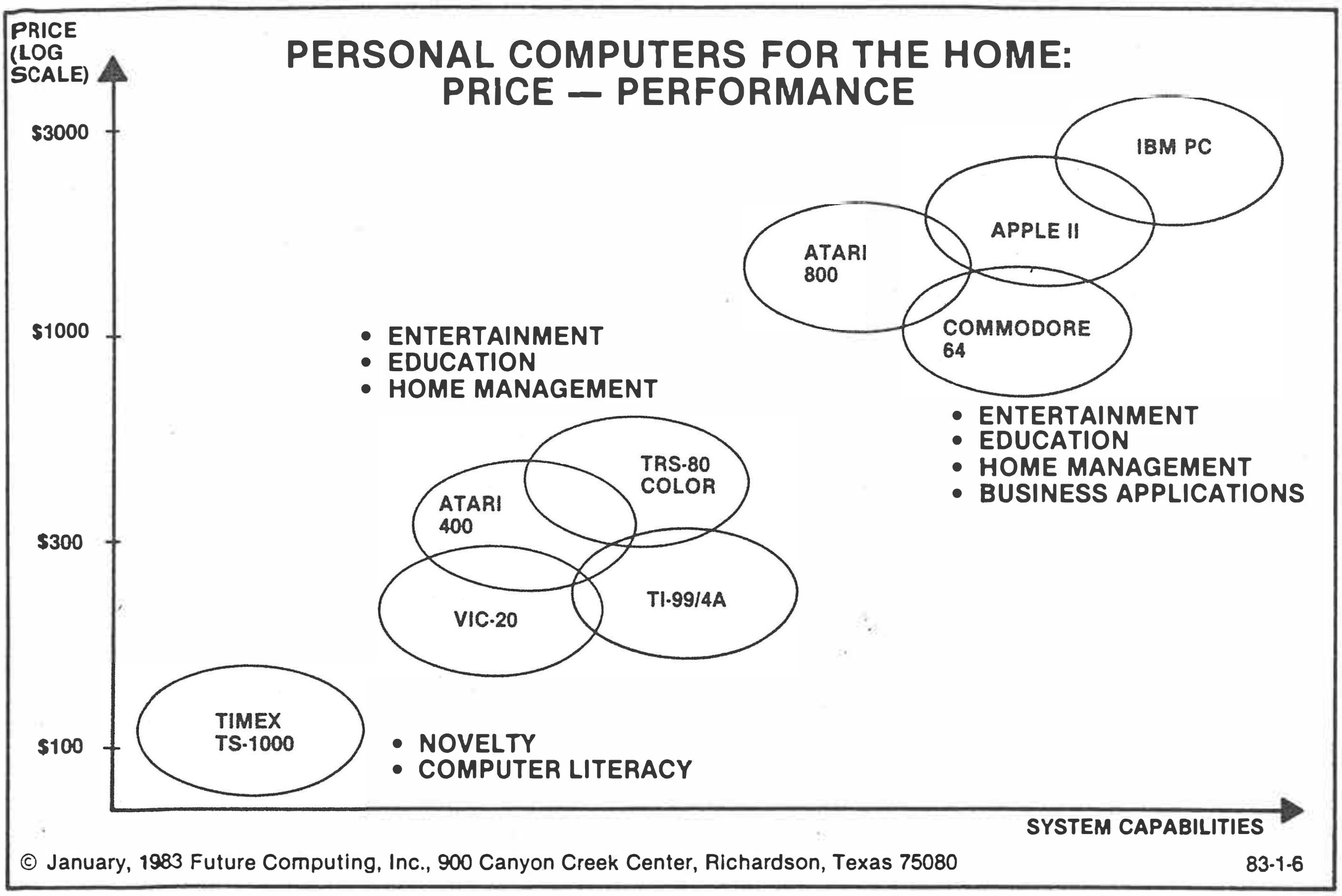
PROCESSOR:	Z80, 3.58 MHz
ROM:	16k
RAM:	6k, expandable by a further 16k
Keyboard:	Rubber keys. 45 keys with auto repeat, contact 'beep'
Mass Storage:	Standard audio cassette recorder 600 baud
Screen:	Television (colour) or monitor, 32 x 16 (text mode), 128 x 64 (graphics mode)
Sound:	Internal speaker
Ports:	Two expansion edge ports, one has full address, data and control lines, the other is just an I/O port
Language:	Microsoft Basic (8k) plus screen, cassette and sound handling (second 8k)

Not Quite Finished Award

Video Technology had a mini-booth, but a maxi-product, the VZ200. The unit has Microsoft Basic in a 12K ROM, 4K

p 40

(cont.)



This chart shows two gaps in the continuum of price and performance of computers, one at around $200 and another at $700-800. The lower gap was totally erased by the new machines introduced at CES. With seven computers under $200 and the announcement of Vic and Atari price reductions, there is continuous overlap from $65 to $600. The price reduction on the Atari 800 and the new Atari 1200XL fall in the upper gap, however, we expect to see more entries before long.

The chart is from Future Views ($365 per year), 900 Canyon Creek Road, Richardson, TX 75080.

of RAM expandable to 64K, eight colors, and one sound channel. Although the screen is medium resolution (128 x 64 pixels), the 64 built-in graphics characters permit excellent graphics to be displayed. A built-in cassette interface and optional Centronics parallel interface help make VZ200 the sleeper of the show at just $99!

If you've been reading *Creative Computing* faithfully, you saw our in-depth review of the Sinclair Spectrum introduced in England about a year ago. Now, **Timex** has brought it to the U.S. as the Timex 2000. It carries a list price of $149 for the 16K model and $199 for the 48K one.

The 2000 is an outstanding computer with 40 real keys, eight-color high resolution display (256 x 192 pixels), ten-octave sound channel (one of us can't hear that much!), upper and lower case, and 16 graphics characters. Our only disappointment is that it does not have a space bar and thus, like the Aquarius, cannot be used for touch typing.

Timex also announced the 2040 printer, a 32-column thermal unit that uses white paper (not the silver stuff of the previous Sinclair printer). It works on both the 1000 and 2000 and costs $99.

At this point it is probably appropriate to announce the

New Computers	Unisonic Futura 8300	Texas Instruments 99/2	Video Technology VZ200	Sanyo PHC20
MPU	Z80A	9995	Z80A	Z80A
Built-in RAM Memory	2K	4.2K	4K	4K
Expandable To	32K	36.2K	64K	16K
Built-in ROM Memory	8K	24K	12K	8K
Type of Basic	Sinclair	TI	Microsoft	Microsoft
Number of Keys	42	48	45	56
Standard Layout?	No spcbar	Yes	No spcbar	Yes
One-Stroke Basic Cmds?	Yes	No	Yes	No
Upper and Lower Case	No	No	No	No
Graphics Characters	20	16	64	0
Text Resolution (Chars × Rows)	32 × 24	28 × 24	32 × 16	32 × 16
Resolution (Pixels)	64 × 48	256 × 192	128 × 64	64 × 64
Colors	B & W	B & W	8	B & W
Sound Channels	1	0	1	0
Octave Range	n/a	n/a	n/a	n/a
Cassette Baud Rate	250	1200	600	1200
Serial Ports (RS-232)	optional	Hex bus		0
Parallel Ports	n/a	Hex bus	optional	0
Parallel Protocol	n/a	TI	Centronics	n/a
Dimensions (Width × Depth × Height)	n/a	n/a	11.5× 6.3 × 2.0	11.8× 6.3 × 1.6
Retail Price	$90	$100	$99	$99

We're Number 1 Award

Three manufacturers tried to lay claim to this award before we even announced it. Commodore, having just produced their 1,000,000th Vic 20 claimed to be Number 1. TI pooh-poohed that and claimed that the 99/4A had made them Number 1. Clive Sinclair was having none of it and claimed that he had been Number 1 for ages. Who is really Number 1?

In terms of sheer number of units, Sinclair is if you add together those sold under both the Sinclair and Timex names (which we think is reasonable to do). If you insist on just one brand, then the Commodore Vic 20 is the leader. By next year, who can say? Maybe TI will claim the Number 1 spot.

IBM, of course, was keeping a low profile. However, we're sure they would insist that dollar volume is a better measure, in which case they are clearly it. Despite having a fair size booth, IBM was not the hit of CES. Quite the contrary, particularly since several trade magazines had predicted that IBM was about to release a consumer computer at CES (they didn't). One even went so far as to put it on the front page of their daily publication on the last day of CES. For this, they and IBM must share the

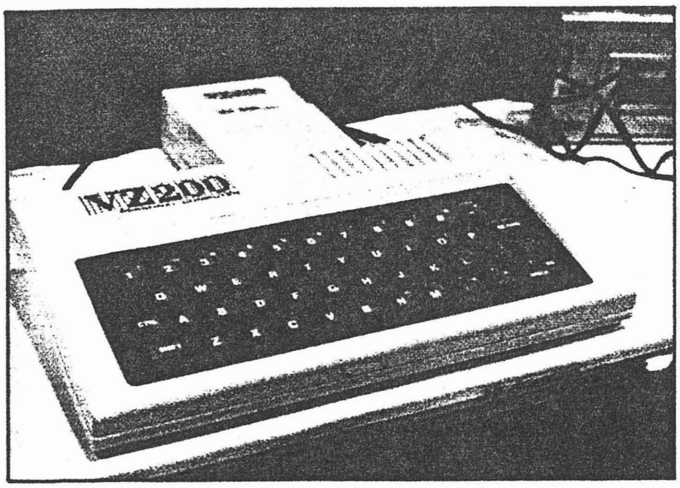
Computer? What Computer? Award

Back to Sinclair printers and peripherals. **Mindware** introduced one of the strangest devices at the show, the Sidewinder, a sideways printer for Sinclair computers. It is also available for the Vic 20, TI 99/4A, Atari and any computer with an RS-232 serial interface.

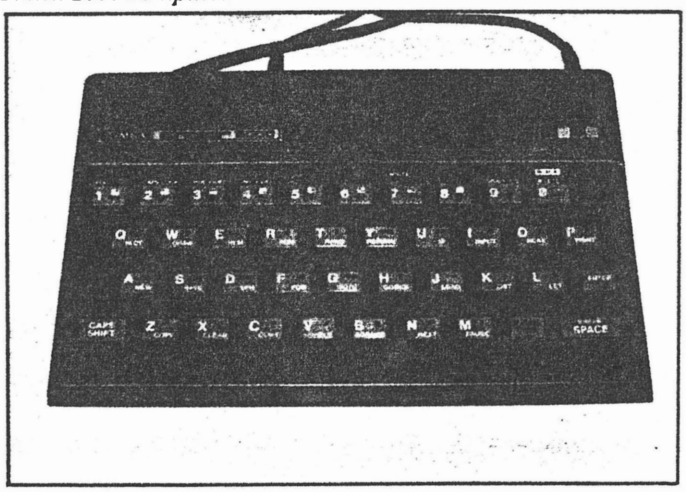
Sidewinder uses 1-3/4" adding machine paper with a dot matrix print mechanism that allows reproducing material wider than the computer display by generating a 12-line printout that runs lengthwise on the paper. Price of the MW-100 is just $139.95.

Data-assette showed several new add-ons and software packages for

Video Tech VZ200 is a great bargain at $99.

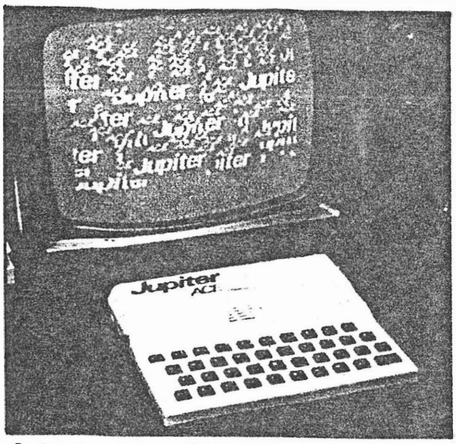


Timex 2000 computer.



2 of 3.

Sanyo PHC25	Timex Sinclair 2000	Mattel Aquarius	Texas Instruments CC-40	Spectra Video SV-318	Panasonic JR-200
Z80A	Z80A	Z80A	9995	Z804	6802
16K	16K	4K	6K	32K	32K
48K	48K	52K	128K	128K	32K
24K	16K	8K	32K	32K	16K
Microsoft	Sinclair	Microsoft	TI	Microsoft	Microsoft
65	40	49	65	71	63
Yes	No spcbar	No spcbar	Yes	Yes	Yes
No	Yes	Yes	No	No	Yes
No	Yes	Yes	Yes	Yes	Yes
	16 (35)	170?	16	52	64
32 × 16	32 × 24	40 × 24	40 × 24	40 × 24	32 × 24
256 × 192	256 × 192	320 × 192	256 × 192	256 × 192	64 × 48
8	8	16	16	16	8
3	1	1 (2 opt)	3	3	3
n/a	10	n/a	n/a	8	5
1200	1500	1200	1200	300/1200	2400
optional		optional	Hex bus	1	optional
1	1	n/a	Hex bus	1	1
Centronics	Sinclair	n/a	n/a	n/a	Centronics
11.8×	9.2×	13.0×	9.5×	n/a	13.8×
6.3 × 2.0	5.6 × 1.2	6.0 × 2.0	5.7 × 1.0		8.2 × 2.2
$199	$149	$200	$249	$299	$349



Jupiter Ace speaks Forth, not Basic.

Timex/Sinclair computers (read all about them in the big *SYNC* directory issue). Also at their booth was the Jupiter Ace computer. While outwardly it resembles a Sinclair with real keys, inside it speaks Forth rather than Basic. Forth aficionados will tell you, usually with no prompting, that Forth is 10 times as fast as Basic, much more compact, and much more powerful. So it makes sense in a small computer like this one (3K).

Commodore was showing several new peripherals, most notably the Vic-1520 four-color printer/plotter with 20, 40, or 80 (tiny) characters per line. It prints sideways or lengthwise on 4-½" wide paper. Price $199. A speech synthesizer spoke to us as we walked by and several new software packages tried to attract our attention as we headed toward the crowd in the back of the booth.

There we found a Commodore 64 redesigned to fit in a portable case about half the size of an Osborne. It had a color display, was battery powered, and looked very inviting. It was just a prototype, but judging from the enthusiasm at the show, it should find its way into production in short order.

Commodore also announced a dealer price reduction on the Vic 20 which should have the effect of lowering the street price to $150, possibly less.

While we're talking about the Vic, we should mention that **Cardco** was showing two expansion boards (one with three slots and one with six), a cassette interface, a light pen, a printer interface, and, hold on to your hats, an adapter to allow the Vic to play Atari VCS cartridges. This latter device was shown with much secrecy in an out-of-the-way hotel room with a rent-a-guard at the door. It gets our

Best Protected Orange Cardboard Box Award

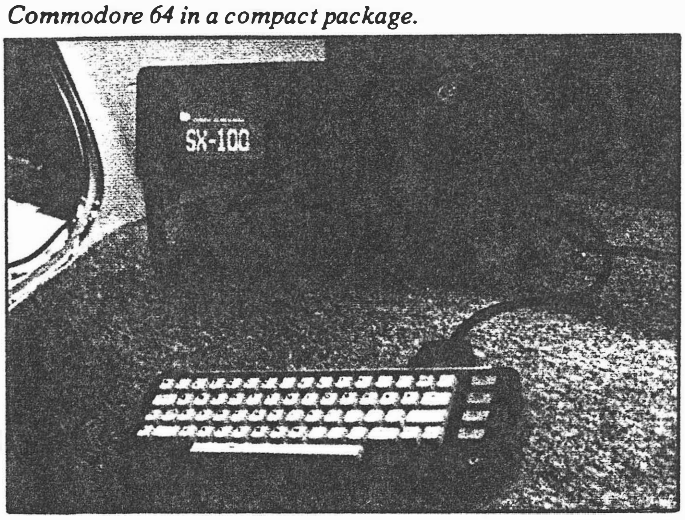
Housed, temporarily we were told, in an orange cardboard and Scotch tape box, the device plugs into the expansion connector on the back of the Vic and has a slot into which VCS cartridges are plugged. It also brings the Vic connector out the back for added memory, etc. The Vic function keys take the place of the VCS switches and the whole thing works like a charm. Price is $89.95.

Spectra Video introduced a new computer, the SV-318, with 32K, Microsoft Basic, CP/M compatibility, 71-key full stroke keyboard, high resolution (256 x 192 pixels) 16-color graphics, and three-channel music synthesizer—all for $299. For this feat, we award them our

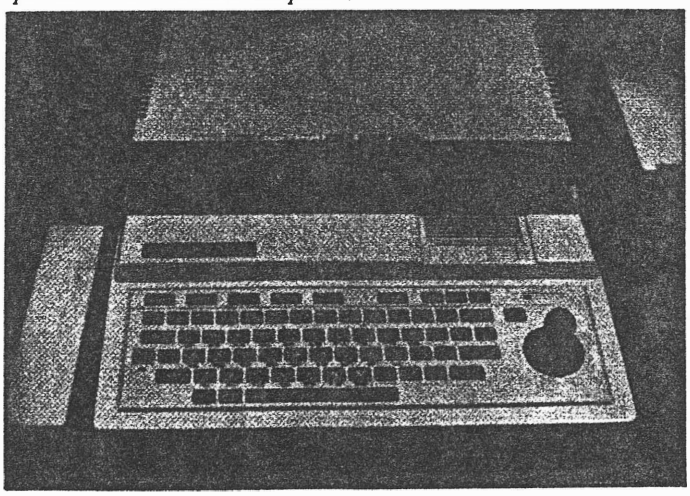
Most Bang For the Buck Award

Not only is the basic computer quite astonishing, but Spectra Video's energetic president, Harry Fox, showed us

Commodore 64 in a compact package.



Spectra Video SV-318 computer.



3 of 3.

creative computing equipment evaluation

Video Technology VZ200 Personal Computer

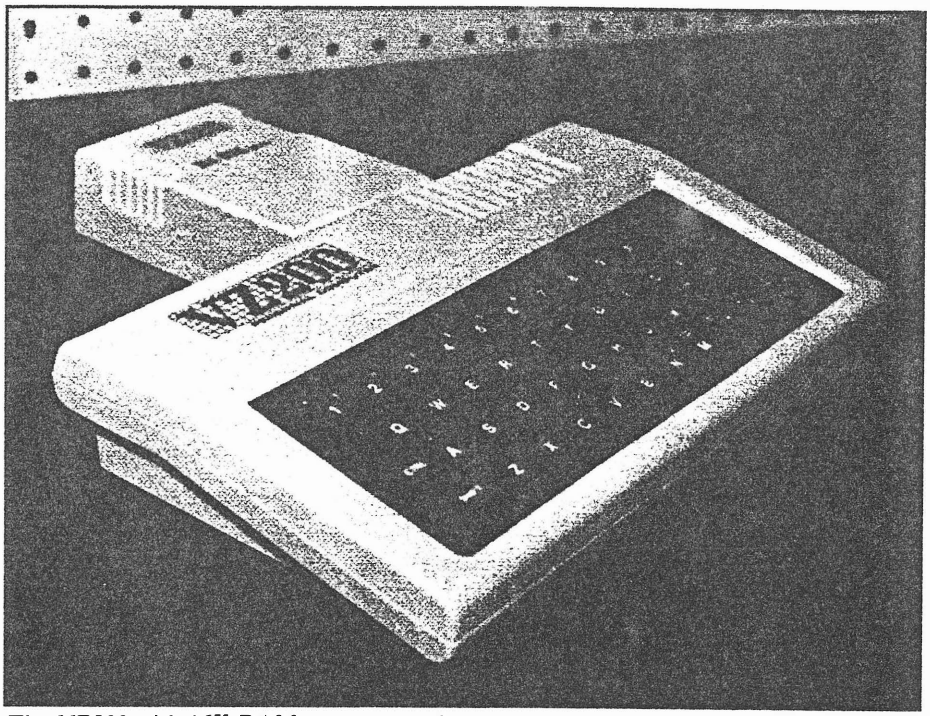
David H. Ahl

The Video Technology VZ200 is a compact microcomputer with a great deal of capability and many unexpected features at a very attractive price.

* The VZ200 is based on the 6502 microprocessor, the same one found in the Apple, Commodore, and Atari computers. The 12K ROM memory includes the monitor and an excellent implementation of Microsoft Basic.

The RAM memory included with the basic unit is a sparse 4K. Two plug-in expansion modules are available, one with 16K and the other with 64K. These modules plug into a slot on the back of the computer and extend out about 5.5".

All the Basic commands, keywords, and functions can be produced with a single keystroke.



The VZ200 with 16K RAM memory pack.

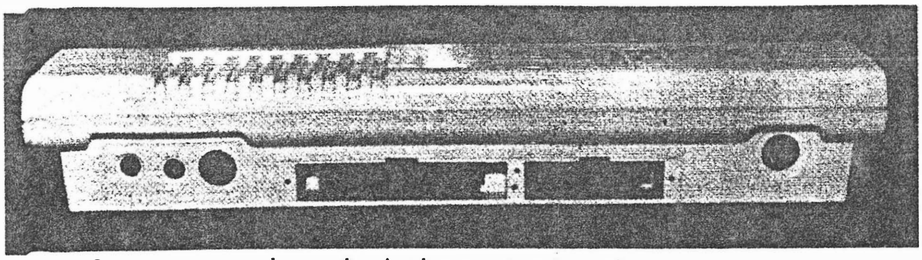
The computer itself measures 11.4" x 6.3" x 2". Two-thirds of the top surface is occupied by a keyboard with 45 keys in four rows. The keys are "Chiclet" style rubber and have a very short throw. Touch typing is possible in only a rather limited way. Although key spacing is the same as on a regular typewriter, the rubberized keys have a different "feel." Much more disastrous for touch typing is the fact that there is no space bar; instead a space key is found at the right end of the bottom row next to the period. This also means that there is only one shift key (at the left end of the bottom row). Several other keys do not have the expected characters; for example the question mark is on the L key.

On the brighter side, each key on the keyboard provides several functions in addition to typing a single letter, number, or character. All the Basic commands, keywords, and functions can be produced with a single keystroke by holding down a control or shift key while the key is pressed. This is very impressive. Most other computers which type Basic keywords with a single keystroke can produce only as many words as there are keys, i.e., one keyword per key. Each key on the VZ200, on the other hand, produces two Basic keywords as well as one or two graphics characters. So each key actually has five outputs: two Basic keywords, two graphics symbols, and an alphanumeric character.

When a key is pressed, it makes a short "beep" indicating one keystroke. If it is held down, it automatically repeats with a beep indicating each key entry.

* Note: Error in CPU type !!

p 26-30 1 of 3



Four I/O connectors and two plug-in slots are on the back.

The top of the computer also has an on/off light. An on/off switch is recessed on the right side of the case.

Peripherals

The VZ200 has an interface to a standard cassette recorder which operates at a Baud rate of 600 bps. This is somewhat slower than other new computers which have rates up to 2400 bps; nevertheless it is twice a fast as machines of just a few years ago. A program that fills the entire 4K of memory with program code takes about 54 seconds to load; a 16K program takes four minutes to load. Bear in mind, however, that most 16K programs do not use 16K of code; much of the memory space is taken by dimensioned arrays and the like.

The manufacturer specifications note that a peripheral expansion bus is built-in, however, we are not quite sure what this means. It appears that expansion modules, which, presumably, can be connected to printers, modems, or other external devices, can be plugged into the back of the computer.

The VZ200 produces two forms of video output: a video signal for a monitor and RF output (on channel 33) for a TV set. It requires 9 volts DC at 800 ma; an AC adapter is included.

Output from the VZ200 can be in one of three modes: text, mixed graphics and text, and high-resolution graphics. In text mode, the VZ200 produces 16 rows of 32 characters (upper case only). Characters can be displayed in regular or inverse video.

Graphics

In mixed mode, text resolution is doubled to 32 x 64 pixels. This is accomplished by dividing each text character into four rectangles. Individual rectangles cannot be addressed. However, 64 graphics character codes define eight characters in eight colors. This gives every combination of the four rectangles in each character. These characters are called with CHR$(128) to CHR$(191). The eight colors are magenta, red, orange, buff, yellow, green, cyan, and blue. If you count black as a color, there are actually nine colors available.

In high-resolution graphics mode, individual pixels can be addressed on a 128 x 64 grid in each of eight colors. To turn on any location, the command SET (x,y) is used; RESET (x,y) turns off any pixel; and POINT (x,y) examines whether a pixel is on or off. Figure 1 is a listing of a simple program that lets a ball bounce around the screen.

```
10 CLS:PRINT "KALEIDOSCOPE   BY
DAVE AHL":PRINT
20 X=1: Y=1: XU=126: YU=62: Z=1
30 INPUT "ENTER 1,2, OR 3";I
40 I=.5*I: J=1
50 MODE (1)
60 X=X+I
70 Y=Y+J
80 COLOR (RND(8))
90 IF X>=XU OR X<=Z THEN I=-I:
SOUND 30,1
100 IF Y>=YU OR Y<=Z THEN J=-J:
SOUND 27,1
110 SET (X,Y)
120 GOTO 60
```

Line	Comment
50	Set hi-res graphics mode
60–70	Compute new x and y position
90–100	Tests to see if edge of screen has been reached. If so, reverse direction of bounce.
110	Draw new spot

Figure 1. Program produces a kaleidoscopic pattern of eight colors on the screen. The input parameter changes the incremental amount added to each successive horizontal or X position. Each time the leading edge of the pattern hits a border of the screen, a beep tone is sounded.

By means of the SOUND (P,T) command, 32 notes or pitches (P) are available which can be played over a wide range of time intervals (T).

On-Screen Editing

Full on-screen editing makes it a pleasure to program on the VZ200. To edit a line of code, it is not necessary to invoke an EDIT command or remember a set of editing commands as one must do on the TRS-80 Color Computer and many others. Instead, on the VZ200, the line to be edited is listed, by itself, with the whole program or with a group of lines. By using the four directional keys on the bottom right of the keyboard, the cursor is moved to the character to be changed. You type the change, move the cursor to the end of the line (remember, a key repeats by holding it down), and type RETURN. Voila! The change is made. On-screen editing can also use the DE-

p 28.

Creative Computing May 83

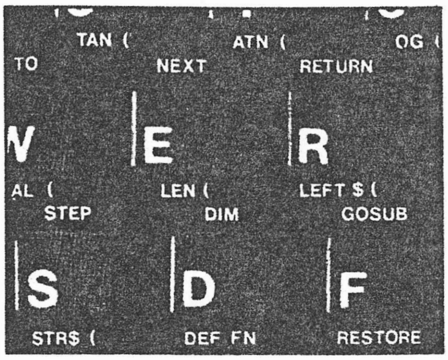
p 26-30   2 of 3.

LETE, INSERT, and RUBOUT keys.

We experienced two small problems with on-screen editing. First, the cursor directional keys are activated by pressing the control key on the left and one of the directional keys on the right. It was all too easy to hit the shift key instead of the control key, but this is probably something that one gets used to after using the computer for a few days. The other problem was that after a while the editing buffer seems to overflow and further editing is not accepted. Admittedly, we were trying to push the computer over the brink and it is unlikely that this will be a problem in normal use.

Problems

Speaking of pushing the computer to the brink, we found several things from which there was no way to recover short of turning the computer off. Even BREAK (the equivalent of RESET on some other machines) failed to return control of the computer to the user. The most common irrecoverable condition was LLIST. This would normally list a program on the line printer. However, if no line printer is attached, the computer hangs. This is particularly bad because the rubberized keys tend to bounce a bit and it is very easy to type LLIST instead of just plain LIST. If you have a long program in the computer and have to turn it off because it hangs up as we did four or five times, you are forgiven if you become a bit surly toward the machine.



Each key produces several outputs.

The surest cure is to use Control/4 to list a program. After a while, we learned to do this.

Other things that would hang the machine are all in the same family, in particular, trying to use a peripheral device that is not attached. In some cases, the VZ200 gave an error message, but in some others it went into never-never land.

We did not have an opportunity to try any of the peripherals. The printer interface module, as mentioned earlier, plugs into the back of the computer. It measures 5.5" x 2" and provides a Centronics parallel signal. The Video Technology printer appears to be a Seikosha unit which we have previously found to be a satisfactory, cost effective printer.

Video Technology also promises a full line of software, however, we will reserve judgment on it until we actually see some of the packages in operation.

Summary

All in all, the Video Technology folks in Hong Kong have done an excellent job producing a versatile small computer. We are impressed with the excellent implementation of Microsoft Basic, full on-screen editing, repeat keys, and easy-to-use graphics features. The idiosyncrasies were a bit annoying, but owners will get used to them and will probably not notice them after a week or two of operation. Bottom line: the VZ200 is a great value for the suggested price of under $100.

Video Technology (U.S.) Inc., 2633 Greenleaf, Elk Grove Village, IL 60007. □

CIRCLE 401 ON READER SERVICE CARD

p 30.

Creative Computing May 83

p 26-30   3 of 3.

New low-cost computer has colour graphics, sound effects

Dick Smith Electronics has introduced a new low-cost personal computer, the VZ-200. Features of the unit include a Z80A processor, eight colour graphics, sound effects, Microsoft Basic in 16K of ROM and both RF and composite video outputs for connection to a standard television set or a colour monitor.

Perhaps the most exciting feature however is the price — just $199, a new low for a colour computer.

The VZ-200 has a 45 key typewriter style keyboard with pushbutton switches (not membrane switches). As the announcement from Dick Smith Electronics puts it "In keeping with the simplified format, the confusing number of switches and controls have been kept to a minimum".

Text is displayed in 32 lines of 64 columns each and graphics resolution is 128 x 64 (horizontal by vertical). Cursor-controlled editing and an inverse video facility is provided as standard and the interpreter allows single key entry of Basic keywords.

As standard, the VZ-200 has 8K of user programmable memory built-in. A 16K memory expansion module is available for $79 which increases this RAM to 24K, plugging into the expansion socket at the rear of the unit.

A cassette interface is standard and a separate printer interface module ($49.50) allows a printer to be connected to the computer.

The VZ-200 is available from any of the 37 Dick Smith stores nationwide.

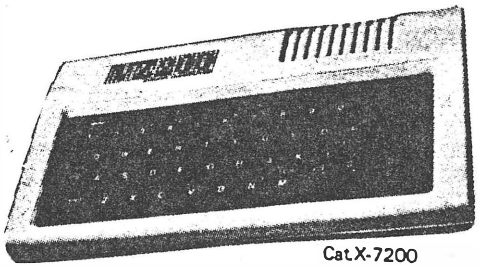
A Colour Computer for under $200!

DICK SMITH VZ-200

pg 4.

First VZ ad. in E.A.

Here it is at last – the personal computer you've been waiting for! With all the right features: colour graphics, sound, standard Microsoft BASIC, both RF and video output, and lots more. Yet thanks to modern technology and Dick Smith's massive buying power, it will cost you only $199 – far less than any comparable machine! Because of this dramatic breakthrough in terms of value for money, our new VZ-200 provides you and your children with the ideal opportunity to learn about computers and programming.

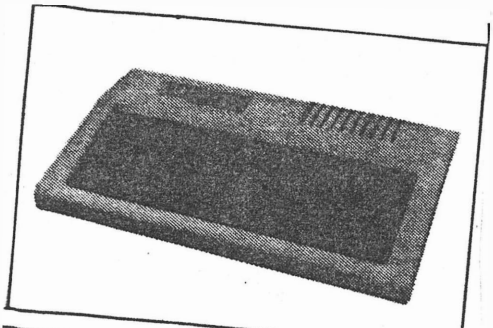


Cat.X-7200

SOME OF IT'S AMAZING FEATURES
- 8K bytes of RAM memory inbuilt
- Powerful Microsoft BASIC in ROM — so your VZ-200 understands easy-to-follow BASIC commands as soon as it's turned on.
- Full colour graphics capability: 8 colours
- 128 x 64 high resolution graphics mode or 64 x 32 — (mixed graphics & text mode)
- Built in sound/music facility: plays any note in a 2-1/2 octave range.
- Large 45-key keyboard, typewriter layout, Auto-repeating keys, too.
- RF output (channel 1) or direct Video output for colour TV monitor.
- Easily expandable using optional 16K memory expansion module, Centronics printer interface module (availability shortly).

AMAZING VALUE

$199



VZ-200

$99 for 8k

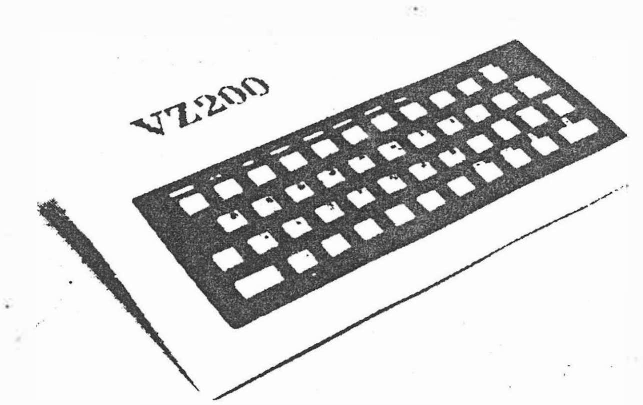
The VZ-200 is the lowest priced home computer in Australia. It suffers from the old problem of awful rubber keys but given its price it's probably not appropriate to complain. When it was launched, Dick Smith (the VZ-200's distributors) expected it to take the country by storm and while many thousands have been sold, it has not attracted the support of many software houses. It's virtually unknown in the US, so you can't look across the Pacific for any third party software support either.

That said, if you're looking for a micro with limited memory but with colour graphics, sound and a reasonable Basic but don't want to spend much on programs (ie, you want to write your own), then the VZ-200's price tag makes it good value for money.

There's not much in the way of being able to expand the VZ-200. Apart from a 'datasette' for $69.50, a 16k expansion module ($79), there's only a printer/plotter and joysticks. There is no provision for disk drives.

If you're into D-I-Y computing, have a look at the VZ-200.

PCG Aug. 84 p 12



Dick Smith colour computer

The Dick Smith VZ-200 personal computer features colour graphics, sound, Microsoft BASIC and both RF and video output.

Priced at only $199, the VZ-200 has been specially designed for the computer beginner. It has 8K of RAM which can be easily expanded to 24K with the addition of the 16K memory expansion module.

The VZ-200 is a fully functional computer, so there's no extra equipment to buy. A comprehensive step-by-step instruction manual is included to teach you how to program in BASIC.

The keys on the typewriter-style movable-key keyboard have been specially designed so that it's difficult to make a mistake. The number of switches and controls have been kept to a minimum.

The VZ-200 has the facility to attach cassette recorders in order to store programs on standard audio tape. The interface module, priced at $49.50, allows the connection of a printer to the computer.

The VZ-200 is now available from any of the 37 **Dick Smith Electronics stores Australia-wide.**

YC Jun 83. p 6.

More Quality For Less Cash

DICK SMITH ELECTRONICS has announced the new Dick Smith VZ-200 colour microcomputer, priced at $199. The system incorporates colour graphics, sound, Microsoft BASIC and both RF and video output.

Features of the VZ-200 include eight kilobytes of inbuilt RAM, eight-colour graphics, 128 by 64 high-resolution graphics mode, 45-key automatic-repeating keyboard, and easy expansion with optional modules. □

Dick Smith's VZ200 personal colour computer

Jamye & Roger Harrison

Since Clive Sinclair dropped his ZX80 and 81 'toy' computers on an unsuspecting and unprepared market, there's been a rush, no — a stampede, to expand the features of personal computers and contract the price. The VZ200 currently sits right at the forefront. What does it offer?

THE VZ200 packs an amazing number of features in such a tiny package: 8K bytes of memory (RAM), 16K Microsoft BASIC in ROM, colour graphics — eight colours in medium resolution and four in higher resolution, programmable sound generator with 2½-octave range and nine different note durations, 45-key moving-key keyboard (with auto-repeating keys), both RF output (to TV antenna input) and direct video (for a monitor), inverse video and on-screen cursor-controlled editing.

The VZ200 measures just 290 mm wide by 163 mm deep by 50 mm high overall. The keyboard is on the sloping front apron and all the attachments plug into the rear. It is powered from a 9 Vdc plugpack. Along the rear apron are the following connectors: dc input socket, cassette recorder jack, monitor output, expansion connector, peripheral connector and TV (RF modulator) output on channel 36 UHF.

The video display only uses about three-quarters of the screen (unlike the picture in the Dick Smith catalogue shows), like many of the colour home computers available. The text format is 32 columns across the screen by 16 lines down. In what they call medium resolution graphics mode you get 64 pixels (blocks) across the screen by 32 down, 128 x 64 (i.e: double) in the 'high resolution' mode.

In the medium resolution mode, you can program a block to be any of eight colours — green, yellow, blue, red, buff, cyan (a blue), magenta or orange. They're what's called the

1 of 3.

'foreground' colours. The background (i.e: the rest of the screen area) can be either green or orange in this mode.

In the higher resolution mode, you can program any block (foreground) to be any of only four colours — green, yellow, blue or red — with the background colour green, or with the background buff you can program the blocks to be buff, cyan, magenta or orange.

The programmable sound generator has a range of 31 notes over 2½ octaves from A_2 to $D\#_5$, plus a 'rest'. There are nine programmable note durations of 1/8, 1/4, 3/8, 1/2, 3/4, 1, 1½, 2 and 3.

The text character set comprises 62 of the standard 64-character ASCII table, 5 x 7 dot matrix format. The two you don't get are hardly important in this application. Thirty of the keys on the keyboard have four 'shift' levels — as can be seen from the accompanying pictures. With the exception of the RETURN, SPACE, CTRL and SHIFT keys, the rest have three levels of shift. That is, apart from obtaining the normal character when you press a key, you can get more functions, such as a graphics character, a BASIC command, an operating command or a program statement.

Four keys act as cursor control keys in the CTRL mode, these being the four on the right of the lower rank. The L and ';' keys provide the INSERT and RUBOUT editing functions in the CTRL mode. The colour programming command keys, 1 to 8, are labelled and colour-coded.

The expansion connector will accommodate such things as a memory expansion module. A 16K module is available for just $79, allowing expansion of the user memory to 24K.

The peripheral connector is for plugging in such things as a printer interface, and one is available for $49.50, permitting the attachment of a standard Centronics printer, many models being widely available — and the prices are continually coming down.

The VZ200 is supplied with all cables in generous lengths, a plugpack, a User Manual, a demonstration program on cassette, a BASIC Reference Manual and a booklet of BASIC Applications Programs.

From the user's view

For all the functions packed into the keyboard, the key operation is a big let-down. The keys are rubber-buttoned microswitches and while they do have movement, the feedback via your finger can only be described as uncertain.

We've criticised this type of keyboard in the past and can't help but think that, where a cost compromise is necessary, an elastomeric keyboard (like that on the ZX81) is preferable. The computer gives a 'beep' when you press a key (except for the CTRL, SHIFT and RETURN keys), which helps, but the key action is so light that double-keying is common. The auto-repeat feature, however, is a good idea. The key will repeat the character or command if you hold it down for longer than one second.

The on-screen editing functions are very good — a real boon to the beginner programmer. The usual BASIC editing feature of simply retyping a crook line works, but that can be time-consuming, especially with long lines. The VZ200 allows you to move the cursor around and re-type incorrectly entered characters, commands or statements. With the latter two, the single-key entry feature is a real time-saver. We would rate the editing facilities as one of the VZ200's major features.

The keyboard has an enlarged SPACE key at the right of the lower rank. This is a problem if you're used to a normal typewriter-style keyboard as you keep cracking your finger on the case below the keyboard! It takes a little getting used to. We also took a little time to learn not to confuse the SHIFT and CTRL keys. There are other problems with the keyboard that relate to its partly non-standard layout, but if you're a beginner in the personal computer stakes it's unlikely to be a worry.

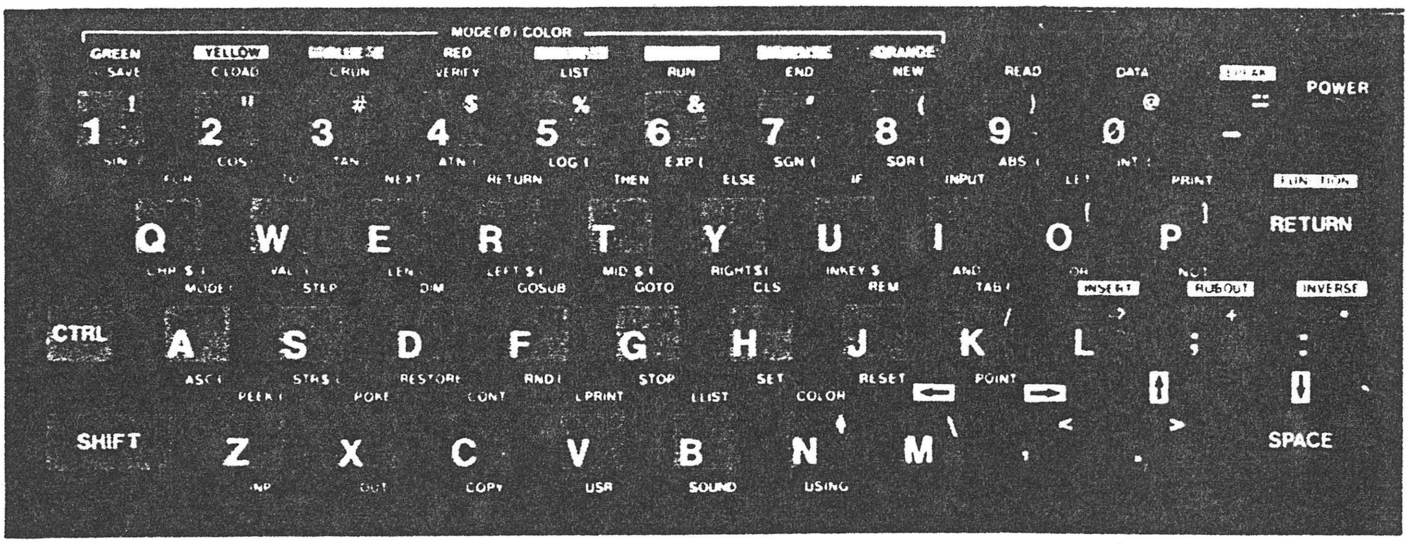
The single-key entering of statements and commands was an idea introduced by Clive Sinclair with his ZX80, forerunner to the ZX81 and Spectrum computers. It's a good idea, taken to its logical limit with the VZ200. Strictly, you need to use more than one key to enter a command, statement or graphics character, but only three at the most; e.g: to get the PRINT command you push CTRL and P together. To get the command or statement under a key, you hold down CTRL and press RETURN, then the key you want.

The direct video output into a Philips 20" colour monitor is good, but plagued by patterning that ripples seemingly diagonally across the display. The display is noticeably inferior when using the RF output into the TV set's antenna. However, it is better than some other popular colour computers around. For the price, it's acceptable.

The VZ200 uses a Z80 microprocessor, probably the most widely used microprocessor in all the personal computers produced to date. The specifications say it runs at 3.58 MHz. However, it's not all that fast, but is probably quite fast enough to manipulate simple graphics effectively.

If you really want to know, a FOR-NEXT loop takes four milliseconds, which in today's computer world is pretty slow. As it really is a beginners' machine, that's no real disadvantage. If you're thinking of ploughing through your maths homework with it, a pocket scientific calculator is faster. ▶

Continued on page 37



2 of 3.

SUMMARY OF BASIC COMMANDS

Functions

1) Arithmetic operators

+, −, *, /, ↑

2) Relational operators

>, <, =, >=, <=, <>

3) Arithmetic functions:

SQR – Square root
INT – Integer part
RND – Random number
ABS – Absolute magnitude
SGN – Sign
COS – Cosine
SIN – Sine
EXP – e^x
TAN – Tangent
LOG – Natural logarithm
ATN – Arc tangent

4) String functions.

LEN – Length
STRS – String of numeric argument
VAL – Numeric value of string
ASC – ASCII value
CHRS – Character
LEFTS – Left characters
MIDS – Middle characters
RIGHTS – Right characters
INKEYS – Check keyboard

5) Logical operators

AND
OR — Relation and logical expressions have value 1 if true, 0 if false.
NOT

6) Graphics and sound functions:

CLS – Clear screen
SET – Plot a point
RESET – Clear a point
POINT – Return the color code
COLOR – Set color
SOUND – Produce tone of different frequency and duration
MODE – Select graphic or text

7) Program statements

DIM – Dimensions
STOP
END
GOTO
GOSUB
RETURN
FOR . . . TO . . . STEP
NEXT
REM
IF . . . THEN . . . ELSE
INPUT
PRINT
PRINT TAB
PRINT USING
PRINT @
LET
DATA
READ
RESTORE

8) Commands.

LIST
RUN
NEW
CONT
VERIFY – Check whether program on tape and memory are equal
CLOAD – Load program on tape
CSAVE – Save program on tape
CRUN – Load program on tape and run
CTRL RESET – To halt program

9) Other Statements

PEEK – Return the value stored at the location specified
POKE -- Load a value into a specified location
LPRINT – Print on line printer
LLIST – List on line printer
INP – Return the contents read from ports
OUT – Send values to ports
COPY – Copy the content on screen to printer
USR – Call the user's assembly language subroutine

Documentation

The BASIC Reference Manual and the two booklets supplied with the VZ200 are generally well produced, clear and understandable — which is just what the raw beginner wants.

The BASIC Reference Manual is spiral bound, which facilitates laying it open so the pages sit flat. However, the spiral binding is just slightly too small for the number of pages and it's a bit of a bind trying to turn them.

This manual covers all the functions and operations of the VZ200 in a fundamental way, with some programming examples. You are encouraged to learn by trying things for yourself. We found a number of small errors, but nothing disastrous.

For example, the method of using the INSERT command when editing does not work the way it's described in the book. Say you typed PRIT instead of PRINT. The book says you do an INSERT by moving the cursor up to the character *before* the place you want to insert a character (that is, 'I' here), type CTRL INSERT, then type the required character (that is, 'N' here). However, that gives you PRNIT!

What you really have to do is cursor up to the character *after* the place where you need to insert a character, then do the insert routine.

The reference manual lists all the available text characters and BASIC statements, operators and commands, with some brief explanations. An error message list is given, but incredibly, no explanation of what they all mean or what to do when you get one! Grrr.

For all its good points, the manual contains no *detailed* index, which would be very useful for a beginner. The contents list is at least comprehensive, so that's a plus in its favour.

What happens when you've worked your way through the reference manual? Well, you won't be a hot-shot programmer, but you will have gained an understanding of programming and be able to tackle some programs of your own invention, plus modifications to published software.

As Microsoft BASIC is used — the erstwhile 'industry standard' — there are huge amounts of published programs and many, many books on the subject that will keep you occupied for ages.

A booklet of applications programs is included with several dozen short programs that are not only interesting and amusing, but instructive and perhaps useful to boot. Many would be good 'starting points' for developing programs of your own devising or useful as subroutines within your own programs.

Absolutely no technical details, not even a memory map, are given, but we guess that such things might appear in some 'support' publications.

The BASIC

The 16K Microsoft BASIC included can only be described as excellent — outshining the mechanical and electronic constraints of the VZ200. But, we have to keep reminding ourselves that this is really a low cost beginners' machine. The range of commands, etc, available, and the flexibility of the language, stand out. Learning to use the facilities is a breeze. The buzzword is 'user friendly'!

All the BASIC commands, operators and statements are shown in the accompanying panel. Those of you who know will see that it's all pretty standard fare. However, it's good to see the inclusion of such things as IF . . . THEN . . . ELSE statements and the COPY statement (otherwise known as a 'screen dump'). Seeing that USR is included for the benefit of using machine code in BASIC programs, we can only hope that some suitable books or manuals on the subject, specifically for the VZ200, will appear at some later date.

Programming using graphics or sound is relatively simple. The graphics commands are simple, largely because of the 'chunky' graphics employed. You'll find no DRAW, PAINT, LINE or CIRCLE commands here, but what you do get is effective for the sort of graphics included in the machine. It's best to crawl before you walk, and it's a beginners' machine, remember. Similar sentiments apply to the sound programming.

Cassette comments

A pre-recorded cassette with cute demonstration software comes with the VZ200. For one thing, it shows that the cassette interface is quite good, as reliable loading was no problem.

As the VZ200 is not a games/computer machine, the pre-recorded software base is only going to be available on cassette, as there's no ROM socket. At present, there's no pre-recorded software available, but, from past experience, that's probably a situation that will rectify itself.

There are lots of 'freelance' software producers in the market supplying software for existing machines who will doubtless get behind the VZ200.

Conclusion

The VZ200 is very reminiscent of the Sinclair ZX81/Spectrum or National JR100 (which is sort of rare here, as yet). It has a very great deal to offer in price, functions and features. The major disappointment is the keyboard, but all low cost home computers compromise here and it's a matter of preference whether you favour one type of cheap keyboard over another.

The big question is, would you do any better at $299. You'd almost certainly get a better keyboard, but we haven't yet seen anything in that price range to compete with the features and memory capacity of the VZ200.

Judging from the phenomenal success and popularity of other 'bottom end of the market' computers, such as the ZX81, Spectrum and VIC-20, there are huge numbers of people who want a low cost computer just to 'get started', or get their children started, in computing.

Price is all-important to people who don't want to pay a great deal of money to learn what the subject's all about before 'getting in deeper'. Compromises are acceptable therefore, and our criticisms should not be taken too much to heart. For its price, the VZ200 has a great deal to offer, and from such small beginnings one can go on to 'conquer the world', or at least a comfortable niche.

3 of 3.

The VZ-200: colour graphics and sound

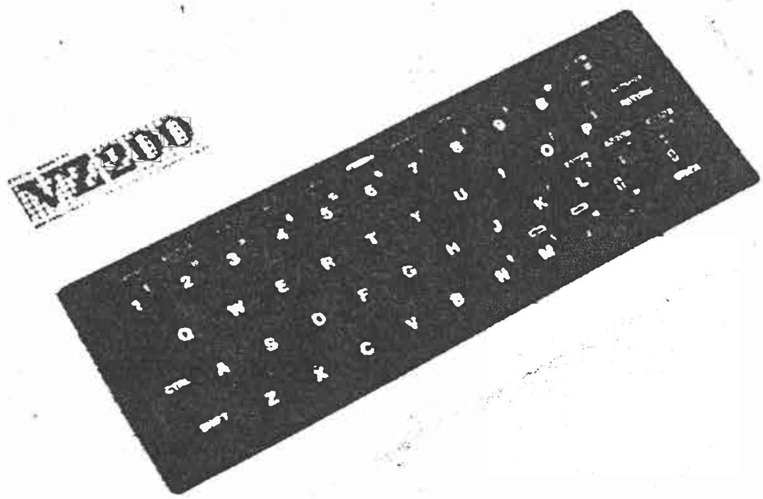
Dick Smith Electronics has done it again with the new VZ-200, a computer with colour graphics, sound effects and built-in Basic for around $200. Others have raved about it, but what's the new machine really like? What does it offer and how easy is it to use?

by PETER VERNON

The VZ-200 computer from Dick Smith Electronics has set a new low price for a colour computer system with Basic. Indeed we can now talk about a class of "under $200" computers, and in this category the VZ-200 is a clear leader. It is the only system for the price that offers colour, a reasonable amount of memory and a powerful built-in Basic interpreter.

With its white case and brown keyboard surround the VZ-200 is an attractive unit. Dimensions are 288 x 162 x 50mm (width by depth by height at rear) with the keyboard sloping to a height of 20mm at the front. There are 45 moving rubber keys but no space-bar as such. A double-sized key at the right side of the keyboard does duty as a space key. All the keys produce an unobtrusive beep, and most serve four different functions.

Pressing a key by itself will produce the character marked on the centre of the key top. Pressing a key in conjunction with "Shift" will produce the punctuation or graphic symbol marked in the upper corner of each key. There are 15 graphic symbols, each a combination of blocks one-quarter the size of a character space. When used with POKE or PRINT@, these symbols allow graphics with a resolution of 64 x 32 pixels in eight colours and may be freely mixed with text.



The VZ-200 computer. The keyboard has 45 moving keys with audible feedback.

Single key entry of Basic statements is activated by the CTRL (Control) key. Pressing a key in conjunction with CTRL will produce the operation labelled on the keyboard above the keytop. Operations handled in this way include cursor movement, insertion and deletion of characters, inverse video and single key entry of about half of the Basic statements and functions. Entering the Basic statements marked below the keys requires holding down the CTRL key and pressing RETURN then the key required.

Although the single key entry of Basic keywords is an advantage, it does require learning key locations and a new typing style which some people might prefer to avoid. An advantage of the VZ-200 is that single key entry, while available, is not obligatory. Statements can also be typed in the normal way, and this may prove faster for a touch-typist. It's nice to have the choice.

All of the keys have an auto-repeat facility, and although it was not mentioned in our preliminary copy of the VZ-200 manual the Basic interpreter supports full-screen editing. Once listed, program lines can be altered by moving the cursor to the position of the alterations and re-typing. When the RETURN key is pressed the alterations will be incorporated in the program. When line numbers are changed in this way the result is a copy of the existing line with the new line number. The old line remains in memory.

The video display

The VZ-200 includes both an RF modulator (VHF Channel 1) and a direct video output, an unusual feature for a low-cost machine. The video display is produced by a Motorola 6847 Video Display Generator chip with additional circuitry to partly adapt the output to the PAL format. The VDG is designed for 60Hz NTSC operation, and the conver-

sion circuitry does not fully eliminate a 10Hz ripple on the screen, even when using a direct entry video monitor.

In the text mode the characters displayed by the 6847 are stable but the sides of the text area show a distracting rippling movement. In the graphics mode the ripple shows up as sideways colour jitter and is most obvious when dots of different colours are displayed in close proximity. This display jitter prevents the VZ-200 achieving the full potential provided by its colour graphics capability.

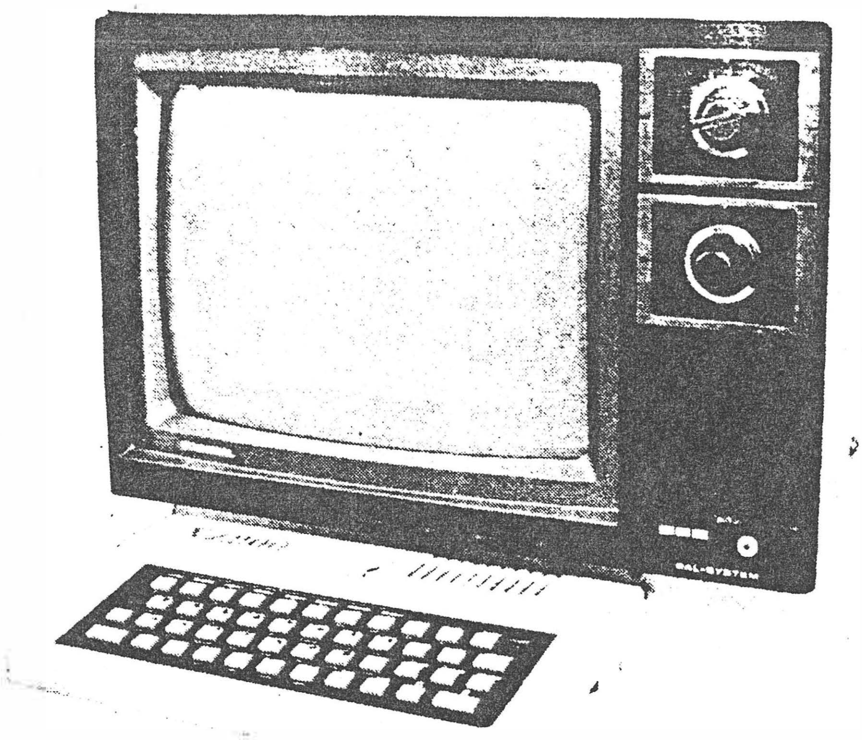
The VZ-200 has two display formats, selected by the MODE statement. In MODE(0) uppercase text only is displayed in 16 lines of 32 characters each, with 64 x 32 block graphics available in eight colours. The normal text display is in light green on a dark green background, but a single Basic statement selects an alternative colour set, producing orange characters on a red background. An Inverse function on the keyboard allows these colours to be transposed to display dark characters on a light background in either colour set.

The statement MODE (1) activates a graphics format which allows plotting on the screen with a resolution of 128 x 64 in one of two sets of four colours each. The COLOR statement selects one of two background colours, green or buff. On a green background the colours available are green, yellow, blue and red, and on a buff background the possible colours are buff, cyan, magenta and orange. Text cannot be displayed in this mode.

Text screens are displayed with a black border surrounding a rectangle of the background colour. On a 34cm (diagonal) video monitor the text display is confined to a rectangle measuring approximately 26cm diagonally in the centre of the screen. MODE(1) graphics are similarly confined by a border, but since the border is in this case the same colour as the background the effect is less noticeable.

The character set of the VZ-200 is contained in the on-chip Read Only Memory of the 6847 Video Display Generator, and does not conform to the widely used ASCII code. Using the same character code with POKE and with PRINT CHR$ will display two different characters on the screen. Presumably software translates between the 6847 codes and ASCII, as statements such as LPRINT and LLIST do work correctly with standard printers.

The Tandy TRS-80 Color Computer also uses the 6847 VDG (although with more extensive modifications for use with PAL displays) and for this reason the text displays of the two machines are similar. Although the 6847 can produce graphics displays in 14 different formats, including 256 x 192 high resolution modes, these facilities are not used by the VZ-200. Most of the VDG control pins are tied to ground in the VZ-200 and there is insufficient memory to support the additional graphics – both situations which could be corrected by adventurous hobbyists.



VZ-200 Basic

Statements and functions of the Basic language of the VZ-200 are shown in Table 1. Numeric operations are accurate within the range 10^{38} to 10^{38} and with the 3.58MHz clock speed of the computer, the interpreter is quite fast. All standard Basic operations are supported, including string handling in the Microsoft format (using RIGHT$, LEFT$ and MID$). A USR statement is included for calling machine language routines from Basic but the VZ-200 does not include a machine language monitor.

In the interests of economical use of memory the VZ-200 restricts the number of subroutines and FOR . . . NEXT loops which can be nested. (A loop is said to be "nested" if it occurs inside another loop, and similarly, nested subroutines are subroutines which are called from within another subroutine.) No more than 30 levels of nesting are permitted in programs for the VZ-200, but this will be found adequate for most applications.

Graphics are handled by the statements COLOR, MODE, SET, RESET and POINT. The statement COLOR I, J will set the characters to the colour represented here by code "I" while in MODE(0) the value of J selects a background/text colour combination, for

VZ-200 Specifications

Processor: Z80A running at 3.8MHz clock speed.
ROM: 16K.
RAM: 8K expandable to 24K with optional cartridge, less 2K for video.
Interfaces: Cassette interface, RF modulator and direct video connectors, I/O connector, expansion connector with full Z80 bus. Optional Centronics type printer interface.
Keyboard: 45 rubber moving keys, most with four functions.
Display: 32 x 16 lines text, 64 x 32 graphics in eight colours, 128 x 64 graphics in two sets of four colours. Inverse video.
Sound: Single voice with 31 frequencies, nine durations.
Software: Basic in EPROM, applications programs on cassette.
Documentation: New documentation under preparation at time of review.

2 of 4.

either a green or an orange background. In MODE(1) the COLOR statement selects one of two possible colour sets, each of four colours, for 128 x 64 resolution graphics.

The statements SET, RESET and POINT are available only in MODE(1). SET and RESET as the names imply turn points on the screen on and off while POINT will return the colour code of a specified point. All three statements require arguments in the form of a pair of cartesian coordinates with the origin of the coordinate system at the upper left corner of the screen. There are no statements for drawing lines or other shapes or for filling areas on the screen with colour.

Sound is produced by software toggling of two bits of an output port driving a piezo-electric transducer in the keyboard unit. Thirty-one different frequencies can be specified, in one of nine durations, with the SOUND statement. The sound is not loud, there is no volume control, and the fixed durations and frequencies limit the sound effects which can be produced. As with colour graphics, however, the VZ-200 scores over its similarly priced rivals which offer no sound effect capabilities at all.

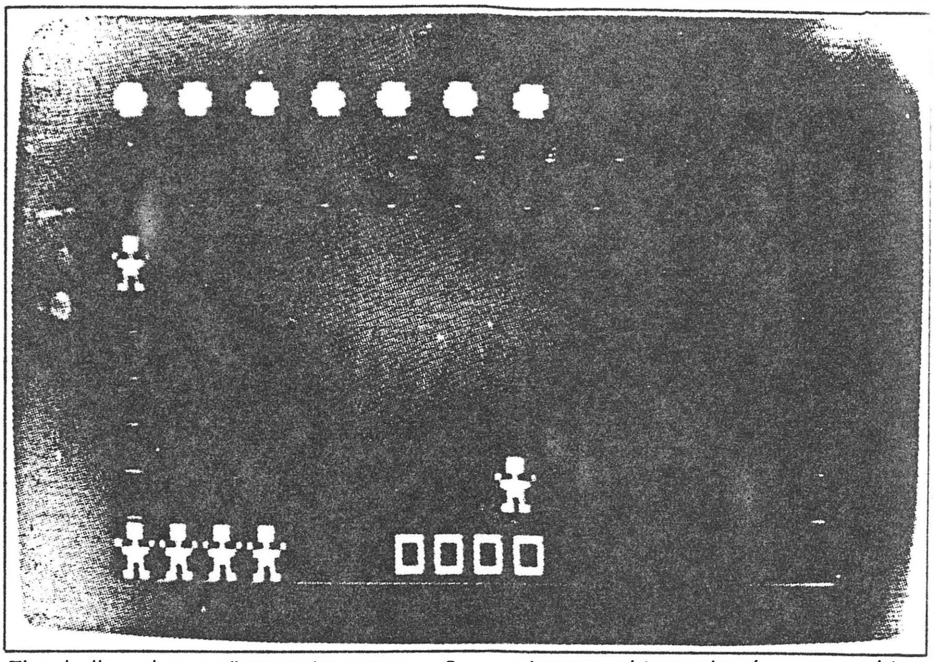
A statement which will be unfamiliar to most is the CRUN command. CRUN, a combination of CLOAD and RUN, allows a program to be loaded from cassette tape and run automatically with a single statement. It is used extensively by the programs on the demonstration tape which accompanies all VZ-200 units.

The cassette handling statements of the VZ-200 also include the familiar CLOAD and CSAVE. Program names can be up to 16 characters long, with the name of each program displayed on the screen as it is found on the tape. The VERIFY statement can be used to compare a program in memory with a program recorded on tape as a convenient assurance of a correct CSAVE, and PRINT# and INPUT# are available for recording and reading lists of data items from tape. We have no information on cassette loading and saving speed but it appears to be around 600 baud.

A COPY statement is also included in the Basic interpreter. According to the manual this statement will copy the contents of the screen to an attached GP-100 dot matrix printer. We could not test this function without the appropriate printer.

Peripherals and expansion

The cassette connection at the rear of the keyboard unit is a stereo socket and the supplied cable terminates in two jacks, one for each for the EAR and MIC connections of a standard audio cassette recorder. There is no motor control of the cassette player.



The "balloon burster" game in progress. Four colour graphics makes for eye-catching games. Over 30 programs are available on cassette for the VZ-200.

At first we had great difficulty in using the VZ-200 with pre-recorded program tapes. Reading tapes we had recorded ourselves was only a problem until we found the correct setting of the cassette recorder volume control.

Using a more expensive National Panasonic RQ-2133 cassette recorder (Dick Smith Electronics, $82.50) however, these problems disappeared and we were able to load all program tapes.

A 16K RAM expansion pack for the VZ-200 is already on the market. This unit plugs into the expansion port at the rear of the machine to provide a total of 24K of user memory at an additional cost of $79.

A Centronics parallel printer interface adapter is also available for the VZ-200. This small unit plugs into the peripheral port at the rear of the keyboard and provides a cable terminated in a standard Centronics type connector. While the Basic COPY statement can only be used effectively with the Seiko GP-100 printer, the LLIST and LPRINT statements will produce text output on any compatible printer.

From the hobbyist's point of view a strong feature of the VZ-200 is the expansion ports provided at the rear of the keyboard. These ports consist of two sets of PCB fingers, normally covered by thin screw-down aluminium plates. One port is labelled "peripheral", and provides access to the Z80 data bus, the lower eight address lines and RD, WR and IORQ control lines, sufficient for the connection of most peripheral controllers, parallel and serial ports etc.

A second port gives access to the complete bus of the Z80 microprocessor and can be used to connect additional memory or memory-mapped peripheral devices.

Some notes on applications

Dick Smith Pty Ltd provided us with a list of around 30 applications programs currently available for the VZ-200. While some of the available games programs

Table 1: VZ-200 Basic statements and functions

ABS, AND, ASC, ATN, CHR$, CLOAD, CLS, COLOR, CONT, COPY, COS, CRUN, CSAVE, DATA, DIM, END, EXP, FOR . . . TO . . . NEXT, GOSUB, GOTO, IF . . . THEN. IF . . . THEN . . . ELSE, INKEY$, INP, INPUT, INT, LEFT$, LEN, LET, LIST, LOG, LLIST, LPRINT, MODE, MID$, NEW, NOT, OR, OUT, PEEK, POKE, POINT, PRINT, PRINT USING, READ, RESET, RESTORE, RETURN, RND, RUN, SET, SGN, SOUND, SIN, SQR, STEP, STOP, STR$, TAB, TAN, USR

3 of 4.

make excellent use of the graphics capabilities and are written in machine language for speed, many of the others can be found in any good book on Basic, without the expense of buying a cassette version.

We also question the choice of some of the programs available. For example, one cassette is a "Portfolio management" program for keeping track of sharemarket transactions. It is unlikely that anyone with sharemarket investments will skimp by buying the VZ-200 to look after them. The two statistics packages may be in the same category — if you want a computer for statistical analysis the VZ-200 is an unlikely choice. If on the other hand you get some statistics problems assigned as homework these two cassettes might be handy to have.

The programs listed in the "Basic Applications" booklet which accompanies the VZ-200 are of the familiar type; sum and average, roots of a quadratic equation, conversion between degrees Celsius and Fahrenheit etc. They serve more as demonstrations of what can be done with Basic on the VZ-200 rather than as serious suggestions for the use of a computer. As such they are a useful tutorial, although most of the programs can be found in existing textbooks. In most cases nothing need be changed to run textbook examples on the VZ-200.

Graphics statements can be added in to take advantage of this aspect of the VZ-200 without difficulty.

Additional programs are under preparation at the time of this review and we expect that independent program suppliers will get into the act as soon as the VZ-200 proves its popularity. Judging from what we have found and the comments of others who have used the computer, this shouldn't be long.

In conclusion

If you want a computer to look after your share holdings, or for word processing, look elsewhere. If, on the other hand, you want a computer for playing games, for self-education, for learning about Basic and perhaps for writing your own programs, the VZ-200 has one overwhelming advantage — the number of features for the price.

If you're handy with a soldering iron and want a computer for taking apart, adding on to and building up, the VZ-200 is also an ideal choice, for the same reason.

The VZ-200 is available from Dick Smith Electronics stores nationwide.

About that keyboard

The most controversial aspect of the key VZ-200, and the one that we found least desirable, is the keyboard. We still can't decide whether it is better or worse than a flat plastic membrane keyboard.

It's not that the keyboard is bad in itself. It's small but the rubber keys move with a pleasant, positive action, and the audible feedback is a great convenience. The problem is that the keys also wobble sideways and back and forward, creating an unsettling effect and, we believe, markedly increasing the chances of typing errors.

Fortunately the single key entry of Basic keywords limits the need for accurate typing, and no one is likely to use the VZ-200 for applications requiring entry to large amounts of text.

We suspect nevertheless that one of the first "add-on" projects for the system will be a full-sized keyboard.

4 of 4.

Laser

The Laser, to be sold through the Computers For All chain of retailers, is the current holder of the title Cheapest Colour Computer. But the £70 price tag has only been achieved at the cost of some pretty drastic corner-cutting.

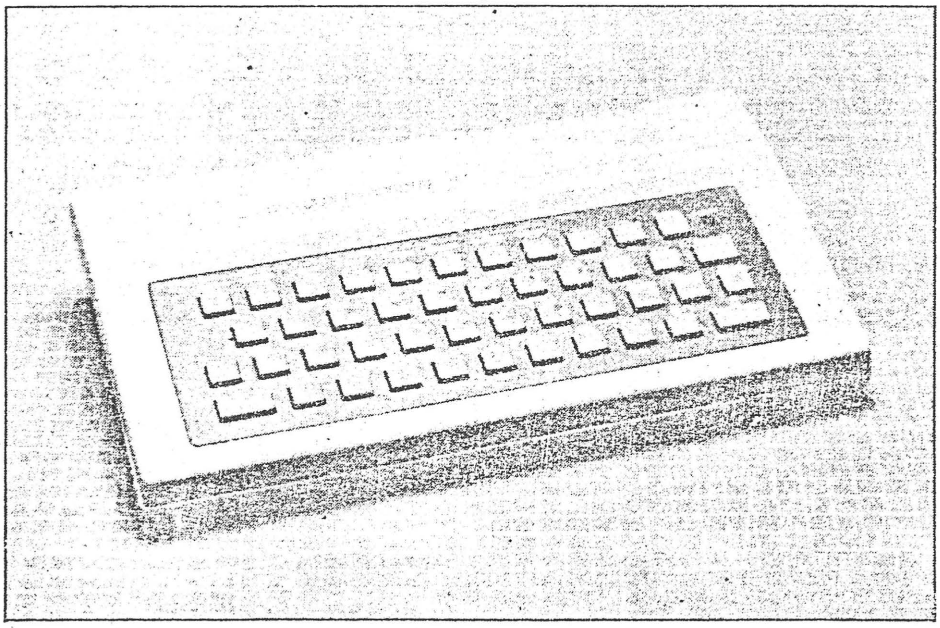
Made by the Hong Kong company Video Technology, it offers a Spectrum-style rubber keyboard, a Z80A processor, 16K ROM and one sound channel. So far so good.

But the graphics resolution is limited to a chunky 128 x 64, and the RAM is downright claustrophobic at 4K.

The Laser boasts one-touch keyword entry, and there are promises of joysticks, a Centronics interface, a four colour printer and a 64K RAM pack. The machine has an uncanny resemblance to the Texet, reviewed in our April issue, and none of the fundamental snags outlined then need to be re-written.

A 16K RAM pack will prove essential for any serious programming, and at £30 this add-on drops the machine into that arena where the Spectrum and Oric are currently slogging it out.

In such company, the Laser's low resolution graphics, lack of established software support, and generally unremarkable specifications are likely to prove fatal.



Timing the Laser's fazer

I'm interested in the Laser 200 which was reported in *PCN issue 5*. You stated that the manuals had to be re-written and the machine wouldn't be released for a month. Since then I've heard nothing about it. Has it not materialised?
Jason Stokes,
Cannock, Staffs

Fear not! The Laser has emerged in the trusty hands of Computers for All on 0286 418414, price £70. Computers for All has also become the main source for that other Hong Kong based machine, the Comx 35, Pro-tested in *PCN issue 15*.

Unfortunately, the same can't be said for the Texet 8000, the Laser's 8K twin. Texet seems to have disappeared for a quick rethink. *PCN* will be Pro-testing the Laser shortly.

Dick Smith VZ200: good value

By ROB FULLERTON

Dick Smith Electronics has released another personal computer on the market to follow closely on the heels of the Wizzard computer. The VZ200 uses a Z80A processor running at 3.58MHz, which must be the fastest clock of all the low-priced personal computers to date.

The computer is quite small, being only nominally larger than the keyboard and 50mm thick. It comes in an attractive white plastic case with the keyboard built into a sloping matte black surround.

A power-indicator LED is the only other feature on the front of the case. An on/off switch is located on the right-hand side. Across the back of the computer there are four sockets for 9v DC power, cassette tape, video monitor, and TV output. There are two edge connectors covered by protective metal plates for the add-on memory expansion and peripheral interface. Power comes from a separate large plug pack rated at 12v 1A. It has a generous length of lead.

Also included with the computer is a lead for connection of a standard audio cassette for program storage and a lead for connection to a monitor or TV. This TV lead is, unfortunately, only long enough to reach to a set placed on the same table as the computer. Other items included in the package are a BASIC reference manual, a book of application programs and a demonstration cassette.

Keyboard

The keyboard is the same used in the Wizzard computer, which is not surprising, since both computers are made by Video Technology, Ltd, of Hong Kong. The moulded-rubber keys are set in a QWERTY arrangement with the standard ASCII character set. Each key performs up to four functions, including the ASCII character screen printed on the keytop, the single key Microsoft BASIC commands, the cursor control, and the on-screen editing.

The alternative functions are accessible by use of the CTRL key in the same manner one would use the SHIFT key. The single-word BASIC commands are printed on the computer above and below each key. All keys except CTRL have an auto repeat facility if held down for more than one second. This is very useful for cursor movement.

Comments I made about the keyboard of the Dick Smith Wizzard computer *(Bits & Bytes, June)*, also apply to the VZ200. The longevity of the screen-printed characters on the keys and the long-term contact reliability of the key switches remain to be proven.

Video display

The VZ200 can use either a colour TV set or a colour monitor for display as both RF and video outputs are provided. The internal RF modulator is tuned to channel 1, Australia, but the picture in N.Z. will come up on channel 2 because of TV channel allocation differences between the two countries. Some re-tuning will be necessary to get the best picture.

The display area for the computer occupies a rectangle covering about two-thirds of the screen. In the text mode there are 32 characters per line with 16 lines displayed. Even with this smaller active display area the characters are sharp and easy to read. The stability of the picture was a little disappointing, however, with persistent diagonal ripples visible on both the TV and monitor displays. The upper-case ASCII character set is displayed and can also be set to inverse video.

Editing

An excellent feature of the VZ200 is the on-screen editing capability. The cursor control keys allow you to position the cursor over any mistake in a line and then, by pressing INSERT or RUBOUT, change the required characters. This saves having to re-type the whole line again as with some computers. The auto-repeat function is very useful here as continued pressing of the RUBOUT will erase as many characters as required. These editing functions rank as one of the most desirable features of this computer, especially for the beginner.

Graphics

Two display modes are available, text mode and graphics mode. In the text mode, the ASCII character set is displayed as well as the 16 chunky graphics shapes. These characters may be displayed in eight different colours with a choice of two

1 of 3.

background colours. For graphics mode the screen is divided into 128 x 64 pixels, each individually addresssable. Each pixel may be programmed on or off with the SET and RESET commands. The pixels may be any of four colours with two background colours. The 8192 pixels displayed in the graphics mode produce quite acceptable resolution for games and data displays.

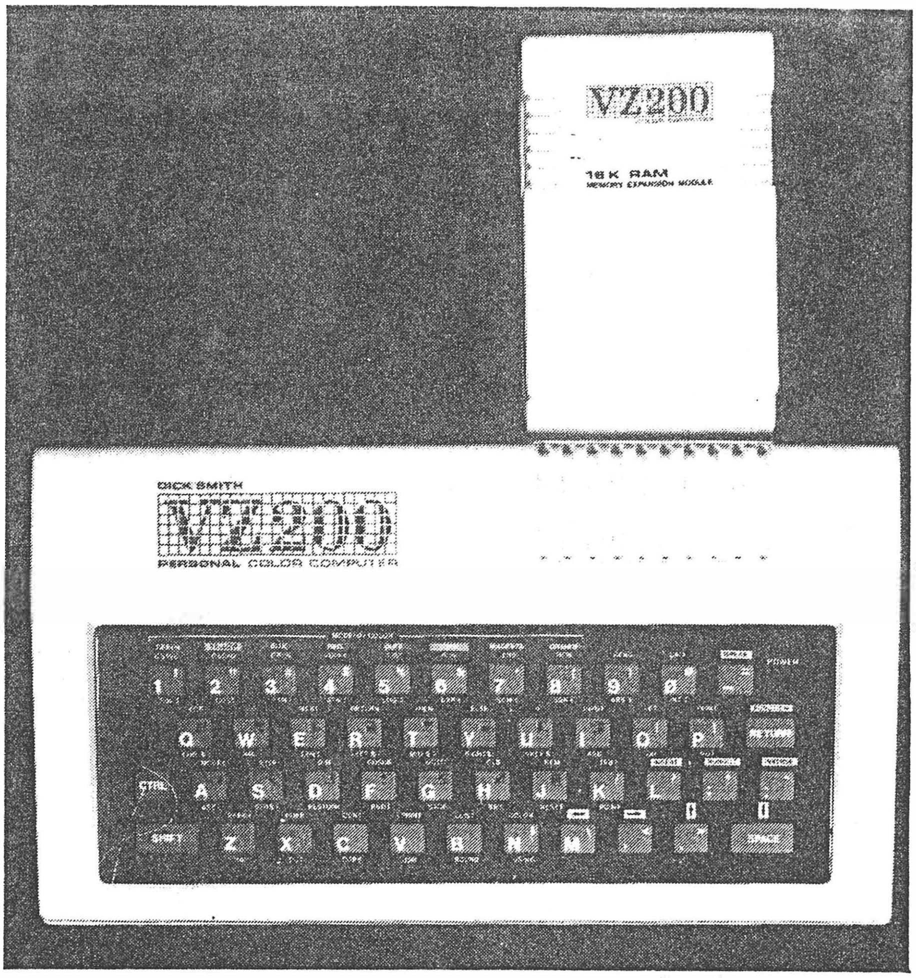
Sound

It is possible to generate sounds on the VZ200 through the internal piezo speaker. Control of the tone frequency and duration is by the SOUND command. Programmable music notes covering 2½ octaves with nine different note durations are available. The sound is very tinny, and with only one channel it can hardly be considered suitable for "serious music programming" as claimed in the advertising leaflets. It is adequate for games only.

Cassette data storage

For program and data storage the VZ200 provides a connection to an audio cassette recorder. An interconnection lead is supplied with two miniature jack plugs on one end and a stereo plug on the other. The stereo plug goes into the computer socket marked tape and the others plug into the ear and mic sockets on the recorder. A demonstration tape comes with the computer which shows off the colour and graphics capabilities.

I found some difficulty in loading this tape as the volume setting for the recorder playback appears quite critical for a successful load. There is no provision for cassette motor control. The difficulty is cured, however, by putting a 15-Ohm resistor in parallel with the earphone connection, i.e. soldering it between the two wires.



The VZ200 with the 16K RAM expansion module

Five BASIC commands handle storage and retrieval of data from the

2 of 3.

cassette. In addition to the usual CLOAD and CSAVE commands there is a CRUN command which works like CLOAD+RUN.

The VERIFY command checks the data on the tape against the RAM data after a CSAVE. This is particularly useful, as the RAM contents are not overwritten and another CSAVE can be given if the first load was erroneous. Files on the cassette are given a 16 character file name, so several files can be stored on a single cassette and the required file loaded by including it's file name with the CLOAD command.

As well as commands to store programs on tape there are two commands, INPUT # "filename" and PRINT # "filename", which allow storage and retrieval of variables and data from within a program that is already executing. The data on the tape is assigned to the variable list given in the INPUT # command. Similarly, the variable list after the PRINT # command is written to the tape. This feature makes a very flexible tape storage system and with a little programming ingenuity multiple mailing list programs and the like should be possible.

BASIC

The VZ200 comes with an 8K version of Microsoft BASIC with 8K enhancements in ROM. This is an excellent version of BASIC for such a low-priced computer and contains many of the features only found on more expensive machines. For example, enhancements such as IF . . . THEN . . . ELSE and PRINT USING are included as well as the USER function for machine code programs.

The BASIC Reference Manual guides the new user through the fundamentals of the language and explains the use of each command with examples. It cannot, however, be considered a serious guide to Microsoft BASIC and a user would have to consult one of the many texts available to obtain the details of the language.

For instance, the published specifications for the ROM BASIC quote single-precision, floating-point maths functions with nine-digit internal precision and eight digits displayed. I found, however, that double precision was available using the D format (eg. 1.2345D+3) instead of the usual E format (eg. 1.2345E+3) and that results can be calculated and printed with 16 significant digits. This suggests there may be other enhancements in the ROM which are undocumented.

Microcomputer summary

Processor:	Z80A running at 3.58MHz.
Memory:	ROM 16K with BASIC interpreter and operating system. RAM 8K (2K screen-6K for user programs). Expandable to 24K, with optional plug in module. Price $149.
BASIC:	16K Microsoft BASIC.
Keyboard:	45 keys in modified typewriter format. Keys auto-repeat after 1 sec. Single key BASIC command entry.
Screen:	Text Mode — 32 char. x 16 lines. Upper case ASCII plus 16 graphics characters for 64 x 32 graphics. Inverse video. 8 colours with 2 background colours. Graphics mode — 128 x 64 pixels individually addressible. 4 colours with 2 background colours.
RF output:	RF modulated signal on VHF channel 2. Cable supplied.
Video:	Composite video 1.4v P-P. PAL compatible. 75 Ohms impedance.
Sound:	Inbuilt piezo speaker. Music notes covering 2½ octaves with 9 note durations. Speaker "beeps" for keyboard entry.
Cassette:	Interface connects to standard audio cassette tape recorder. Data rate 600 baud. Cable supplied.
Power supply:	Plug pack. Output 10v DC at 800mA.
Manuals:	User manual, BASIC Reference Manual, book of sample programs.

Memory addresses for the video portion of RAM are given for text and graphics modes. This enables PEEK and POKE to be used for direct screen addressing in graphics and games programs. The INKEY$ command, which polls the keyboard and returns the key value if pressed or a null string if no key is pressed, is a further feature which enhances games software. It is unfortunate that a complete memory map is not included.

The greatest feature of a computer with Microsoft BASIC is the enormous range of software written in this "industry standard" language. There are many books of programs written for Microsoft BASIC including those for the TRS-80 and the System 80. These should provide the VZ200 owner with an extensive software library to adapt to his computer.

Memory expansion

The memory of the VZ200 can be expanded from its internal 8K RAM to 24K with the addition of the 16K expansion module. This plugs into the rear of the computer in the appropriately marked socket. It is a rather bulky package which relies only on the edge connector for physical attachment. If the computer is to be lifted it would seem wise to unplug the module before moving to prevent undue strain on the connector.

The other connection at the rear of the computer is available to accept an interface for a Centronics-type printer. This interface, with printer cable attached, is obtainable from Dick Smith for $99. The Microsoft BASIC provides good software interface for a printer as the LPRINT command can be used with the USING command to give formatted printing. As well as the LLIST command is a COPY function which allows the screen contents to be dumped to the printer.

The expansion of the VZ200 is not limited to a printer only. The product leaflet quotes joysticks, games cartridges, larger expansion memories and serial and floppy disk interfaces as "coming soon".

Summary

For the first-time computer purchaser the VZ200 offers excellent value for money at $349 for a complete up and running system. The 16K Microsoft BASIC interpreter has many enhancements not found on other personal computers in the same price range. The single key BASIC commands and on-screen editing make it an ideal machine for learning to program. The memory expansion to 24K and a printer interface make the VZ200 a powerful performer. The keyboard is definitely a disappointing feature, however, this should not prevent the prospective first time computer purchaser from giving the VZ200 very serious consideration.

3 of 3.

YC Aug 83.

Cash & Carry Computers

1 of 9.

– or, The Slashers Strike Again!

As prices drop in the home computer market, the competition is hotting up. Here Les Bell looks at the latest crop of under-$500 machines to see what's what ...

TWO YEARS AGO, we ran a story on 'Slashing the Cost of Home Computing', as we were amazed at the price reductions and new low-cost computers that were appearing. Well, we're still amazed. It is hard to believe that you can get so much real computing power for so little money!

Of course, at these low prices, you don't get – for example – the mass storage facilities afforded by disk drives. By and large, your mass storage takes the form of cassettes. However, disk drive controller cards are starting to appear as an option on an increasing number of small home computers.

Virtually all these machines have colour displays and will accept plug-in games cartridges, so they will do double duty as educators and entertainers. When the kids get bored with creating coloured shapes, they can always blast away at some for a while!

Another new feature that has started to appear only recently is the built-in joystick, mounted next to the keyboard. This often doubles as a cursor controller. I doubt that it will be long before low-cost home computers are supplied with a mouse!

Anyway, read on, and compare these little tykes with your $10,000 S-100 boat anchors. Eat your heart out, Altair!

Microbee IC

If you've been in Australia for a while and haven't heard of the Microbee, you haven't been reading the papers or magazines or watching television. The Microbee was launched on an unsuspecting world in the February 1982 issue of *Your Computer*, and proved to be an enormous success. Initially available in kit form at just under $400, it has since been upgraded and improved, repackaged and generally changed into a bigger computer than it used to be.

The Microbee IC is the latest incarnation of the little mite, and offers the most popular enhancements and options, together with a few improvements, all in one package. The IC is faster (3.375 MHz clock) than earlier models, and incorporates as standard both the WordBee word processor ROM and the NETWORK communications ROM.

The IC uses MicroWorld Colour BASIC V5.22, which includes additional commands to set the foreground and background colours and modes. Thirty-two colours are available for the foreground, not all of them describable, and eight for the background.

Listings can now be set to be in either upper or lower case, according to the user's preference; typically, I find lower case easier to read.

It seems that Applied Technology is planning to release more software in ROM form for the Microbee. Up to (theoretically) 256 different ROMs can be plugged in, and the command *PAK n* will select the appropriate ROM pack by outputting the value of *n* to the memory bank select port.

In the IC, two ROMs are provided as standard. Most useful probably is the WordBee version 1.2 ROM. WordBee is loosely modelled on WordStar and Electric Pencil, and incorporates a surprising number of useful and powerful commands for such a small system. Version 1.2 contains several new features, such as the ability to vector output to one of a number of outputs, which gets round a major problem for many Microbee owners. In addition, touch typists can select input from an external keyboard which they may prefer.

Other new WordBee commands include underlining and double striking, and a new command allows the user to move the cursor to the end of the current line.

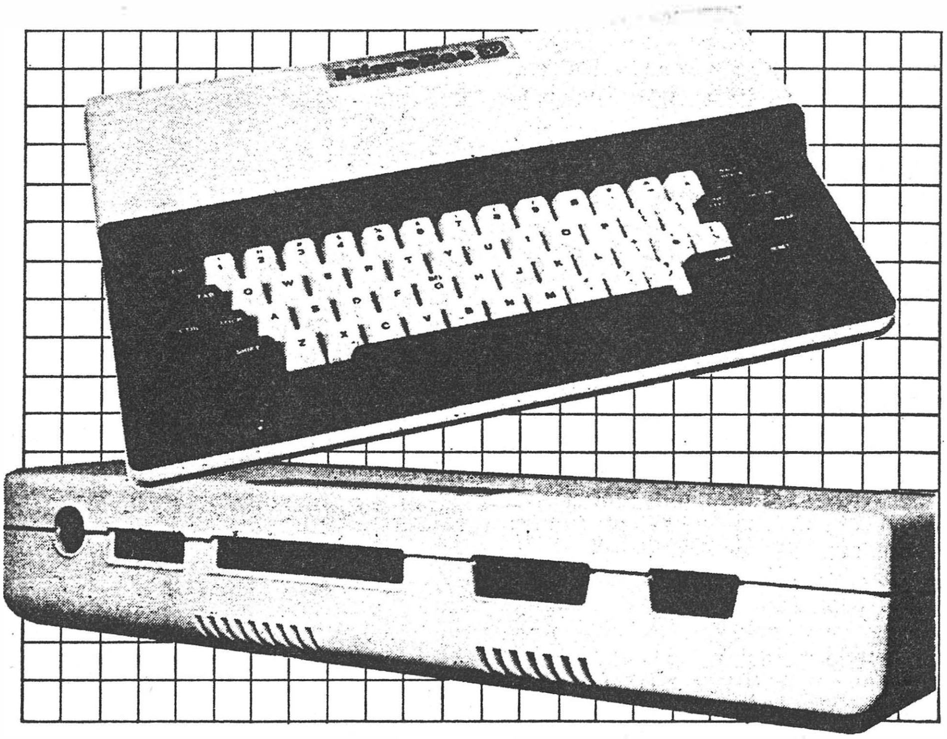
The other major addition is the terminal/network ROM which not only provides communications facilities, but provides a number of other general tricks accessible from BASIC or elsewhere. The general NET command will turn the Microbee into a full or half duplex terminal with an 80 by 24 screen which emulates most of the codes of the Televideo 912 terminal.

The baud rate is settable at 110 to 4800 baud, and parity can be odd, even or off. Best of all (to us here at YC particularly) the NET ROM implements file transfer using the Christensen protocols, so that Microbees can now communicate with each other and the popular bulletin boards.

The network ROM is accessible from BASIC or from within WordBee, providing a range of extra communications and screen formatting options.

The documentation for the Microbee is continually improving, and the latest versions of the user manuals are very good indeed. The Microworld BASIC manual is well organised for both tutorial and reference use, and is quite readable.

The Microbee has always been a powerful and capable little computer, but this latest version really is a winner. Its design is oriented towards useful activities, such as word processing and communications, rather than game playing – but a heap of games are available if you want them!



YC Aug 83 2 of 9.

Dick Smith VZ200

At just under $200, Dickie's come up with another winner here. The VZ200 is a neat little computer indeed.

The VZ-200 is virtually a totally non-technical machine for the user who wants a gentle introduction to BASIC programming and home computing. For example, nowhere in the manual does it say what kind of processor is under the hood! Indeed, there is virtually no technical detail at all anywhere in the manual.

All this is possibly to the benefit of the completely non-technical novice who could do without that kind of intimidation. But it bodes ill for the future availability of professionally written games and utility software. I'd say that for the near future at least, and excluding whatever Dick Smith may release, the VZ200 will remain a BASIC-only machine.

The VZ200 is probably based on the ubiquitous Z-80, and is supplied with 8 Kbytes of RAM as standard. A 16 Kbyte memory expansion module is available for $79.

The BASIC interpreter used is, of course, Microsoft's Extended BASIC, complete with colour graphics and sound commands. The screen displays 16 lines of 32 characters each, and the keyboard is a calculator-style QWERTY with a soft action. Like most of the machines covered, the spacing between keys was less than I would have liked; obviously they are designed for somewhat smaller fingers than mine.

Two graphics modes are available: in mode 0, the graphics resolution is 64 by 32 pixels with nine colours available and text displayable. In mode 1, the resolution is 128 by 64 pixels in eight colours, and this is a better mode for games and more complex graphics.

The graphics statements are the standard kind used in the TRS-80 Colour Computer and other machines with Microsoft Colour BASIC. A point is set with the statement SET (X,Y) and turned off with the RESET (X,Y) statement. POINT(X,Y) will return true if a point has been set and false if it has not. The colour is set using the COLOR statement, which sets the foreground and background colours.

The background can be either green or orange in mode 0, while in mode 1 only four colours can be selected for each background colour.

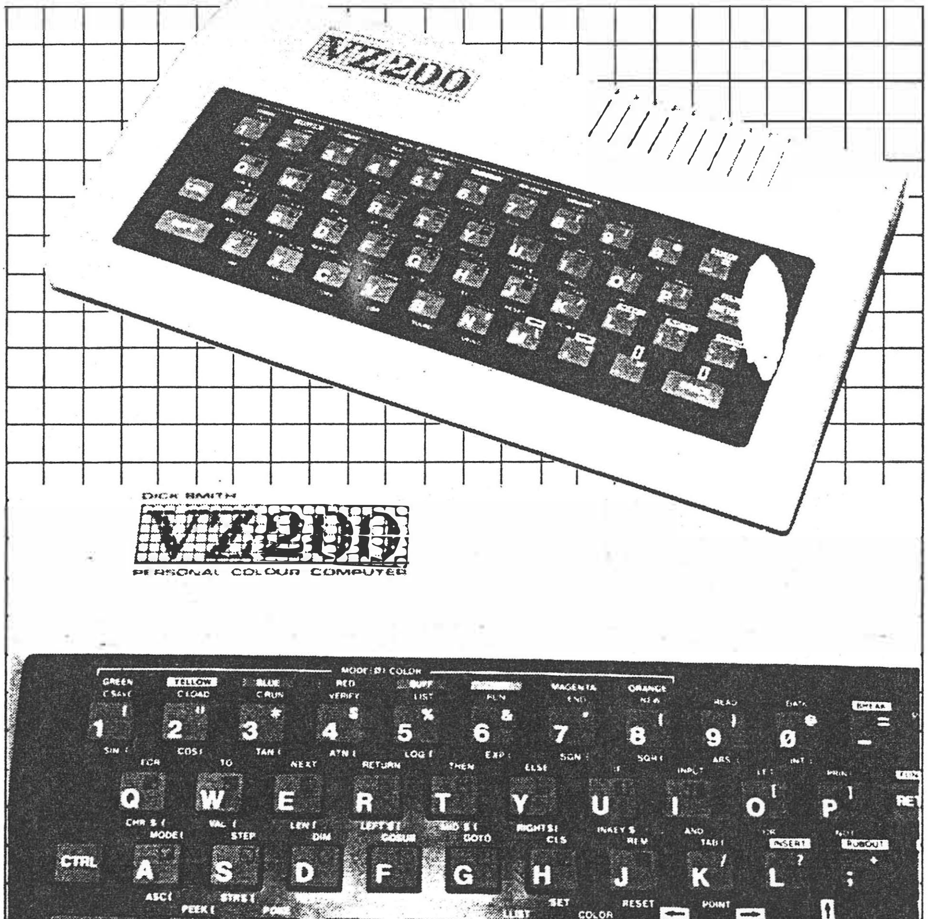
The SOUND X,Y statement will generate a tone of pitch X and duration Y. By using data statements, it is possible to create quite complex little tunes.

For those who want to dabble in some PEEKing and POKing, the manual does give the addresses of the screen RAM, so some fast updating can be done that way, though this will require some experimenting.

The manual is well written, and is organised as a tutorial text, bearing in mind the likely audience for this kind of machine. There are no signs of the Janglish that usually mars manuals on this kind of machine.

Expansion is limited on the VZ-200; there is a socket on the back for the plug-in 16K RAM module, and a peripheral connector, obviously intended for a printer. Apart from the cassette cables, that's it. For parents who don't want their kids to get carried away buying more and more extras, that's probably a blessing!

While this computer probably won't do much for the dedicated enthusiast who wants to get into machine code programming and interfacing all kinds of peripherals, it's just right for those who want to learn some programming and not get bogged down in unnecessary details. Run a business it won't; draw you in to the joys of programming it will!



YC Aug 83 3 of 9.

Spectravideo SV-318

This little package starts off with quite a small personal computer, but it is expandable into a full computing system of quite useful proportions.

The basic console, which contains the computer proper, is only a few inches deep and not that much bigger than, say, the Sinclair machines – but it includes 32 Kbytes of ROM and 32 Kbytes of RAM as standard. The keyboard is a calculator type with a soft feel but a reassuring amount of travel, and is easy to use.

The keyboard includes all the standard QWERTY characters, including the tilde and the escape and control keys, but it also includes five shiftable function keys (which are pre-programmed for BASIC keywords) and some miscellaneous keys for functions such as character insertion and deletion. In addition, the 71-key keyboard is marked with a set of graphics characters.

An unusual feature of the keyboard is the joystick/cursor control pad at the right, which can either be used as a cursor pad for editing, or with the joystick plugged in to double as a games controller.

Inside the box there's a Z-80A microprocessor running at 3.6 MHz, with 32 Kbytes of ROM containing Microsoft BASIC and 32 Kbytes of RAM, half of which is dedicated to graphics. The ROM is expandable to 96 Kbytes using plug-in cartridges, while the RAM can be extended to 256 Kbytes.

The graphics capability of the SV-318 is impressive. The screen resolution is 256 by 192 pixels, with 16 colours available. Most important to games creators, however, are the 32 sprites which are available. These movable shapes can collide with each other and other objects, or can pass in front of or behind each other.

The SV-318 also provides three sound channels, fully controlled by the built-in BASIC, which will allow the user to write music or provide background sound effects for games. The sound circuitry is capable of background operation, so that the BASIC can continue the action in games while a sound is being synthesised.

A major plus of the SV-318 is its expandability; a range of plug-ins and accessories is available that would make many other manufacturers green with envy. These range from a wide selection of games to an expansion chassis which will accept an RS-232 port, Centronics interface, extra RAM, a disk controller and 80-column card. Other options include a graphics tablet, games keypads and joysticks, data cassette recorder and dot matrix printer.

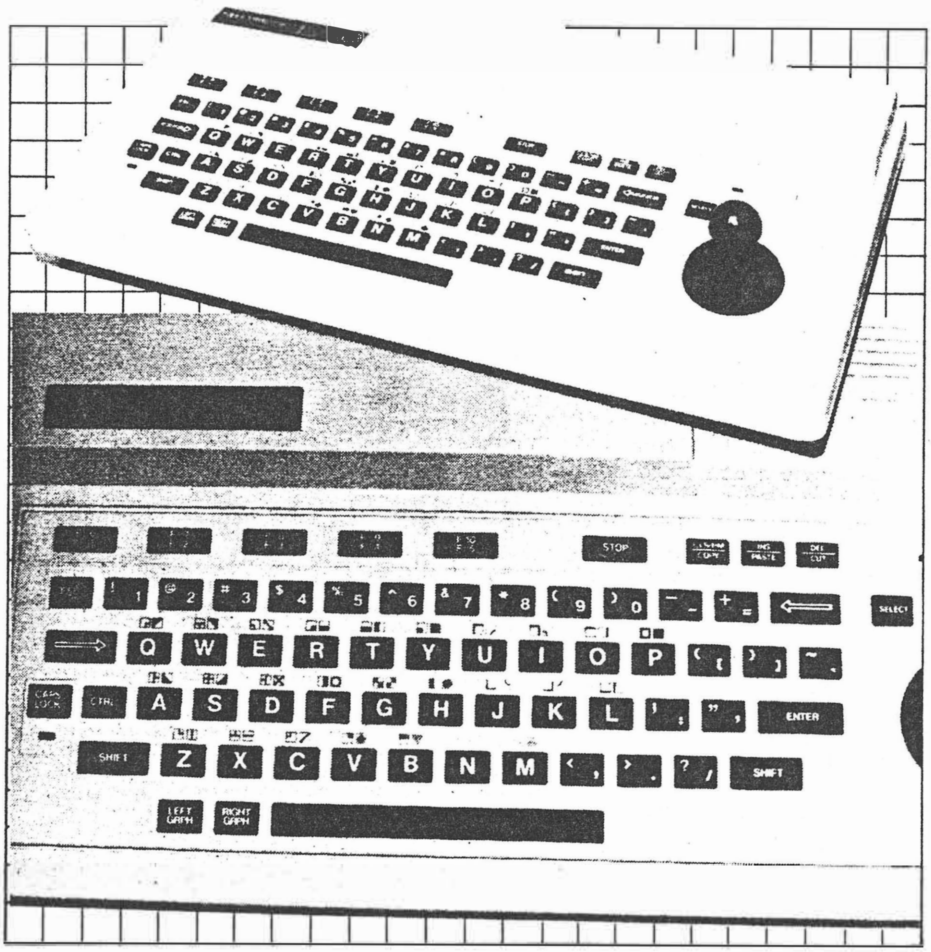
When used with the optional disk drives, the SV-318 uses the CP/M operating system, so that a wide range of software is available. The disks have a capacity of just over 160 Kbytes (formatted).

Setting up the SV-318 is just a matter of plugging in the power supply and modulator and wiring it to the back of the TV set. Immediately you're up and running. Attaching more of the peripherals and options might lead to a rat's nest of wiring, but this isn't a problem providing you don't want to move it all too often.

The SV-318 has an extended version of Microsoft BASIC, which includes all the usual graphics commands and the music macro language similar to that found in the IBM PC. It also includes provision for interrupt handling.

The manual for the SV-318 is nicely produced, and teaches the user BASIC through the use of the graphics statements – certainly a more interesting method than the usual mathematical approach. It progresses nicely until the end, when more complex statements, such as the SPRITE$ statement, are demonstrated by example but not explained. The user is left to try to deduce how the statement works.

The SV-318 is a nicely put together system; it has enough expansion capability to satisfy a wide range of requirements, and would be a good choice for the person who knows he wants a personal computer, but can't decide what for . . .



YC Aug 83 4 of 9.

Micro-Professor II

There has been much talk recently about so-called 'rotten Apples' and Apple look-alikes. We have generally kept quiet on the subject; the legal complexities of registered designs make it a tricky subject, and even where such designs are legal, we feel it is very much a case of *caveat emptor*. These machines may suit some buyers, but in general, would you buy a car which claimed to be a copy of the Holden Commodore yet sold for only $3000?

One machine which seems to be able to stand on its own merits, yet offers a high degree of Apple compatibility, is the Multitech Micro-Professor II. This little (175 by 240 by 30 mm) box contains a 6502 microprocessor with 64 Kbytes of RAM and 16 Kbytes of ROM containing a monitor program and BASIC interpreter which accepts Applesoft BASIC programs.

The screen display looks just like the Apple's and the memory map (where the ROM fits, and where the graphics memory is) also looks just like that of the Apple. The Micro-Professor II can read Apple cassettes and, with the addition of a disk drive, floppy disks.

On the top front of the box is a 49-key calculator-style QWERTY keyboard, which – as with most of these keyboards – will horrify a touch typist. The tiny keys are closely packed with a non-standard layout.

The left-hand side of the case has a connector for a games cartridge, a Centronics printer port and a remote keyboard or joystick port. At the rear is a 50-pin connector which looks remarkably like an Apple expansion slot, but is subtly different and could not be relied on for full compatibility.

The major use of this slot is for the Micro-Professor II disk controller card, which connects the disk drive to the computer. The disk system is supplied with an MDOS II disk operating system, which bears more than a passing resemblance to Apple DOS 3.3 – but is obviously different in several ways.

The Micro-Professor II display is composed of 25 lines of 40 characters each, upper case only, just like the original Apple II and II+, and it provides lo-res and hi-res graphics in just the same manner. However, it is not completely Apple-compatible, and some mention of the differences may be in order.

While the Micro-Professor II can accept Applesoft programs, many such programs perform PEEKs and POKEs of memory locations associated with the monitor and graphics routines. These locations are different on the Micro-Professor II, and so such programs will not work, as a rule. However, if you understand the purpose of these PEEKs and POKEs, you can probably rewrite the program to work on the Micro-Professor II.

When it comes to machine code programs – the vast majority of good fast action games – the situation is even worse. These programs always use monitor routines and I/O port addresses to perform their I/O, and there's no way you can find those references and change them to work on the Micro-Professor II. The original author, who has the source code, could do it – if he thought it was worthwhile.

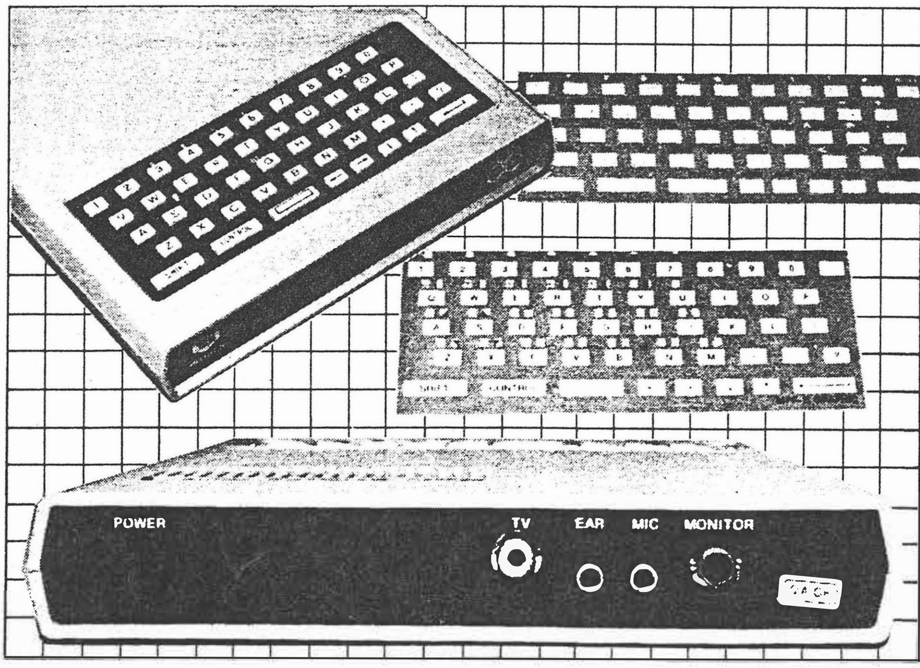
In summary, the Apple compatibility of the Micro-Professor II is probably of most use to someone who has already acquired a lot of experience with the Apple and is fully conversant with its operation. Such an individual could probably rewrite his/her own software to run on the MP without much difficulty. There's certainly not enough information in the Micro-Professor II manuals to get along without some of the Apple documentation as well, particularly when trying to use the graphics features such as shape tables.

In short, less than 100 percent compatible is not compatible; being slightly incompatible is like being slightly pregnant. And of course, where Apple is concerned, being 100 percent compatible is dangerous!

Someone should tell Multitech that quantity of documentation is no substitute for quality. The User's Manual and Introduction to BASIC Programming are reasonably well organised and take a good stab at providing plenty of reference material and technical background. The trouble is the translation into decidedly non-idiomatic English.

Thus we are presented with interpretation problems: what does the translator mean? What is a 'straight-thinging person'? How about 'Even if I were presented with a computer as a gift. I would be troubled as to whether I had enough room in my study to for it'.

The Micro-Professor II is, nonetheless, an interesting little machine which offers the ability to take advantage of the huge amount of software published in magazines and books. It also offers good value for money. It can run some quite good games cartridges and be a lot of fun – but I certainly wouldn't try to replace an Apple as a small business machine with a an MPF-II.



Y.C. Aug 83 5 of 9.

COMX 35

The COMX 35 is a very interesting machine indeed. For one thing, it's the first personal computer (other than simple single-board types) I've come across that uses the 1802A microprocessor. For another, it doesn't use Microsoft BASIC, but one that I've never seen before.

The COMX 35 has 16 Kbytes of ROM containing BASIC, and 32 Kbytes of user RAM, plus the screen RAM of 3 Kbytes. The display is 24 lines of 40 characters each, and eight different colours are available. The COMX display is unusual, in that computer output and echoed user input are displayed in different colours. As the user's manual points out, this is a useful feature for beginners.

The COMX 35 keyboard is a rubber type with a spongy feel, and a slightly non-standard (but still basically QWERTY) layout. At the right side of the keyboard is a built-in joystick. The computer has a built-in sound synthesiser and speaker.

Most micros use Microsoft BASIC; it's almost a novelty to find one that doesn't. COMX 35 BASIC is rather unusual. It is based on the ANSI standard, but with several extensions. Interestingly, it's an incremental compiler design. This means that when the user types RUN+, the interpreter does a scan through the program source code, and replaces all jumps to line numbers with jumps to an absolute address in memory.

This means that the program will run significantly faster, as much of a conventional interpreter's time is spent searching for the next line to be executed. Of course, if a program is edited, all the absolute addresses are changed, and so it will return to normal operation next time the program is run.

Graphics control is available in the COMX 35 through a user-definable character set, using special BASIC statements to manipulate it. This allows complete control over shapes and colours, including the creation of multi-colour shapes. The accompanying blurb also stated that the COMX 35 had "enhanced graphics developed along the Logo language" which I presume means turtle graphics, but I could find no mention of this in the manual and wasn't able to try it.

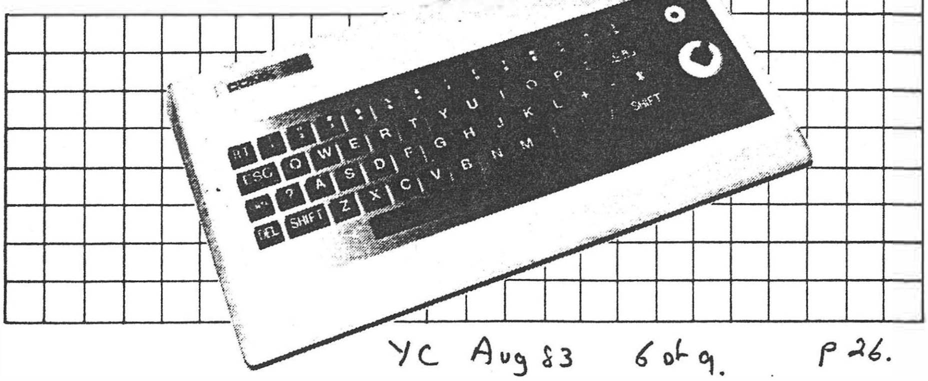
Other features of the language include the ability to set timer interrupts – great for game design – and the ability to save the entire data area of a program onto cassette. This presumably does away with the need for data files, although it means that data sets are restricted to the size of memory.

For those who find BASIC a bit limiting, the COMX 35 will also run Pascal or FORTH, so collectors of linguistic esoterica will be happy.

Various options are available for the COMX 35, including plug-in ROM packs, RS-232C and parallel printer interfaces, and a disk controller and drives. A speech synthesiser is also available.

Many programs are available for the COMX 35, including an electronic spreadsheet, simple databases, financial and statistical functions, a range of education programs and, of course, games – heaps of them – such as Othello, Hangman and various shoot'em-up, eat'em or catch'em variants.

An interesting machine, this; perhaps it will appeal best to the buyer who is happy to write his/her own software and will never want to key in programs straight from magazine pages without conversion. It's certainly an interesting one...



YC Aug 83 6 of 9. p 26.

Commodore VIC-20

Well, what can one say? This is the machine that really turned Commodore around; it was getting a bit staid with the old PET series of machines, but the VIC really breathed life into the company.

The VIC has 5 Kbytes of memory – not a lot, these days – and can display up to eight colours. The memory is expandable to 32 Kbytes, using an external expander, and there is a plug-in ROM socket at the back.

Perhaps the best feature of the VIC is its full-sized keyboard, which even a fussy typist like me finds enjoyable to use. The keys are labelled clearly with alternative meanings like the graphics shapes and colours. At the side is a games port for a joystick, while the rear of the case boasts a row of cassette, user, serial and video ports.

The range of plug-in cartridges for the VIC is tremendous, and they really show the high-resolution graphics capabilities of the machine off to good advantage. The user can get at them with a super expander cartridge, which gives a 176 by 176 resolution.

The VIC's sound effects set a new standard in their time, with three voices of music. The whole package was quite revolutionary, and has done well in the intervening years – just look through the ads in this magazine . . .

Tandy Color Computer

This beastie reached our shores in late 1981, and provided a look at an alternative way of putting together a home computer. The Color Computer boasts a resolution of 256 by 192 under Color Extended BASIC, with nine colours, and as with the VIC, some of the games really make good use of that display.

The standard amount of ROM is 16K (with Extended BASIC) and most machines will have either 16 Kbytes or 32 Kbytes of RAM. The microprocessor used is the Motorola 6809E, an exceptionally powerful chip that is used to do a lot of the legwork inside the CoCo and save on hardware costs.

The CoCo can be expanded with a disk system, and many TRS-80 Model I programs could be run on it with virtually no modification, so it has quite a lot going for it.

The CoCo never really caught on in a big way (probably because the VIC was just so much cheaper), which is a shame as it really is a nice machine. It has a dedicated user group which does all kinds of weird and wonderful things with it and it really has a high level of support behind it, both from Tandy and elsewhere.

Texas Instruments 99/4A

The 99/4A is rather a problematical machine. It's almost sent Texas Instruments broke – figuratively speaking – yet it is a nice machine with all the things we are told a home computer ought to have.

The 99/4A has a 16-bit processor (though its BASIC is unaccountably slow) and a special graphics processor chip which looks after the 256 by 192 graphics and drives the sprites (moveable shapes) around the screen. The 'A' model features a decent keyboard compared to the earlier model.

A user-definable character set allows the user to create chess pieces, card symbols and other shapes.

The TI machine's manuals are well written, and TI arranged for additional material to be published by traditional publishing companies. Perhaps the greatest effort has been poured into software, with the release of a vast range of games, home applications and utilities, including TI Logo, a full implementation of that most interesting language, which really takes advantage of the graphics capabilities.

Other options include a speech synthesiser, expansion box, memory, disk drives and various I/O boards.

The machine has a definite following, with a large national user's group which has branches all over Australia. It's well supported and has great appeal.

 YC Aug 83 7 of 9.

Sinclair Spectrum

Worthy successor to Uncle Clive's ZX-80 and ZX-81, the ZX Spectrum is one of the tiniest personal computers around. It's hard to obtain in Australia, as the United States market apparently has first claim on production, but hopefully the situation will ease with time.

The Spectrum has a rubber-type keyboard, which has so many functions, symbols and letters it can be rather confusing at first. And, of course, it has a colour display, a major leap forward over the old '81. Eight colours are available, in two intensity levels.

For those who keep filling memory with their programs, good news: the Spectrum has 16 Kbytes of RAM as standard, and that is factory upgradeable to 48 Kbytes. The BASIC interpreter is in a 16 Kbyte ROM.

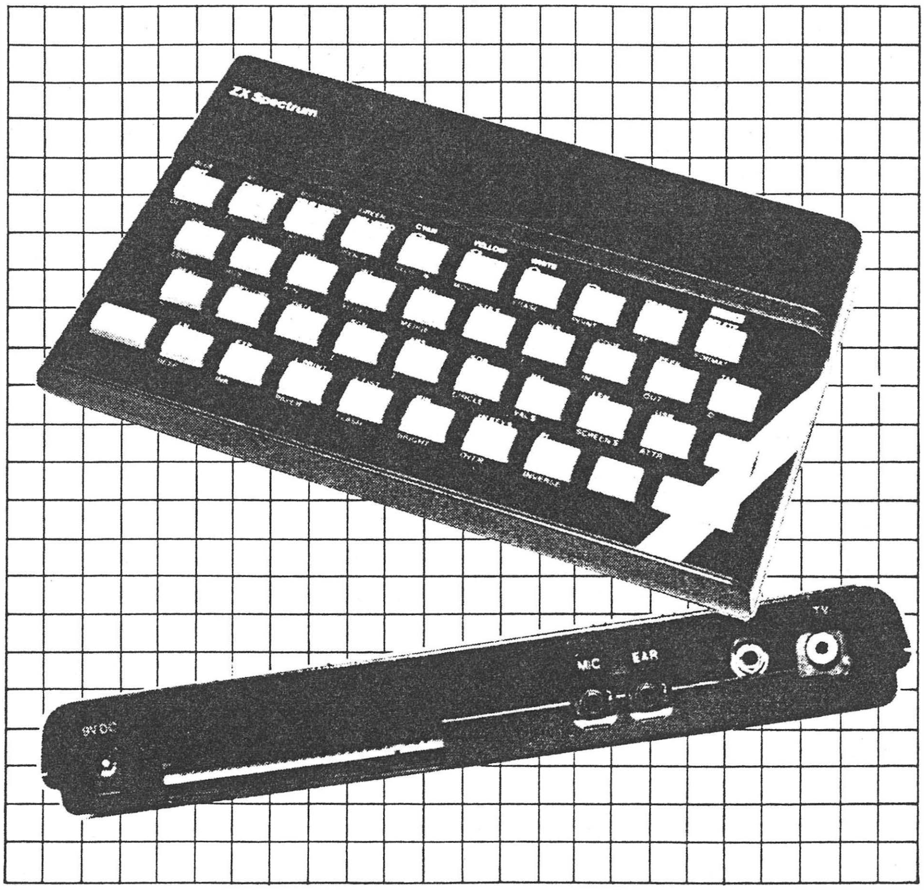
The Spectrum BASIC is a superset of the ZX-81 BASIC, with some additional statements. These include statements to change the colour of the border, the background and the foreground, as well as invert the colours, flash, and draw lines, circles and arcs.

The user can define his own character set by using the BIN statement, which allows him/her to specify which points to turn on in an eight by eight matrix.

The biggest let-down about the Spectrum is its sound capability – or lack of. It can synthesise a single tone through software, which of course stops the action while it makes sound effects.

The major form of mass storage is a cassette, but low-cost 'micro-floppy' drives are available – in fact these use a stringy-floppy style of tape wafer. An electrostatic printer is also available, which can print graphics off the screen.

The Spectrum has achieved remarkable success in Britain, where it seems almost everybody has one. Because of this, there is a fantastic amount of software appearing for it, plus books, magazines and the general support a home computer owner needs. It's going to be a successful machine.



Not being a great game player, Les sought out an expert to deliver judgement on Dick Smith's Wizzard games machine. He dragged Mark Burnicle out of the pub for long enough to write this report . . .

SOME TIME ago, at a somewhat hard to remember party, Les Bell turned to me and said something about me writing an article for *Your Computer*.

Upon my enquiring on what the heck I would be able to scribble about, Les muttered 'Wizzard'. This brought an instant quizzical look from me as I'd been called many things, but wizzard – never!

'No, you can review the Wizzard!' he beamed at me, presenting me (almost magically) with a black cardboard box about the size of a beer carton and splashed with colour. Although not having a clue what I was holding, but not wishing to appear ignorant, I returned his excited banter about the black box with the pictures of Dick Smith, and muttered the word 'Wizzard' several times.

Eventually all appeared fine so, black box tucked safely under my arm, I set off to (as it turned out) review the Dick Smith Wizzard.

Once out of its package the Wizzard takes on the appearance of a quite simple, compact unit, complete with two joysticks, touch pads and firing buttons.

I decided, in a fit of irrational confidence, to connect the cords to their various points; but when all my efforts had failed, I reverted to the instructions and was soon under way.

To become accustomed to the machine I opted for a familiar invaders-style game called Sonic Invader. Since I was quite proficient at this style of game and had kept one particular hotel in Ultimo from going out of business with my patronage of its machine I decided this would provide a good idea of the Wizzard's ability.

To warm up I decided on battle number one of 16, a singles game which is Invaders at its most basic. The invaders move slowly from side to side, dropping slow moving tracers at you as they get cut to shreds. Definitely a good warm up. So on to battle number five, another singles battle – this time, however, the little devils kept disappearing. This was a bit tougher but I still managed to rack up points without much trouble.

At this point, full of confidence, I tried number 13, the hardest singles battle. The invaders move very quickly, rain you with bombs and disappear. This took a few goes before I managed to get the idea, but I soon had them under control.

I was impressed. The graphics were as good as the pub and milk bar versions, the controls moved quickly and accurately and it performed as well as you could hope for.

The doubles games were extremely challenging, with your opponent firing at exactly the same time as yourself

Having warmed to the controls and the enjoyment of the competition, I tried the next game, Planet Defender, a variation on another of the standard games found in pubs and the like.

It Gets Harder . . .

Planet Defender finds you appearing from hyperspace to confront aliens which bear a remarkable similarity to bats and Halloween pumpkins. The game itself is a step up in ability and reflexes. Not only do you get to eradicate these little nasties on sight, you also plan your defence with a 'radar' device at the top of the screen. This is a game which tests the reflexes to the limit at the maximum difficulty stage.

The multi-colour bats and the green Halloween pumpkins glide in and out and then engulf your craft until you disappear in a thousand little pieces. A good game with very good graphics, the normal invaders type sound effects – though on the Wizzard these don't sound tinny but clear and crisp.

Finally (around 2 am the first time I played) I tried Tennis. This one really got me in. No longer is a video game of tennis restricted to two hyphens moving back and forth preventing a full stop from going past – no way! Things have progressed slightly.

I switched on the set and fell off my chair (*drunk again, Burnicle* – **Ed**) – there before me was a tennis court complete with net, crowd, Wimbledon sign (why settle for less?) and two little tennis players complete with racquets who run around actually hitting the ball over the net. The graphics are so damn great that the ball even casts a shadow.

The games got progressively harder, faster and more frustrating. You may compete against another player or (if you like a challenge) against the machine. Believe me, the machine likes to win.

Playing an early level game I had it under control: serve, volley, volley again and the machine hits the ball right into the net. So, feeling capable of wiping McEnroe off the court I progressed up a few grades and followed my successful game plan. Serve, volley, volley again, and the machine hits a top spin lob over my player. While the early games play fairly predictably, the higher level games appear to hit at random patterns.

This game will turn you on (for hours), it will enthrall your little sister, your big brother, your mum, your dad and anyone else you may happen to show Wizzard Tennis to.

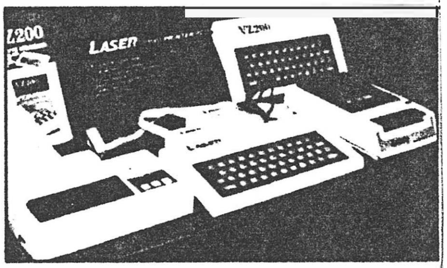
But be warned, if you sit down and try to beat the machine before you go to bed, you'll be there for a while. You will need a good supply of Scotch, a very comfortable chair and the patience of a saint.

This one will have you playing for hours. It's a Wizzard. □

9 of 9. YC Aug 83 p 32 & 33.

Speaking of low end, **Video Technology**, previously in the very low end with handheld games and the VZ200, was showing some up scale computers and a very nifty four-color printer/plotter for the VZ200. Incidentally, in the rest of the world, the VZ200 is known as the Laser computer and it is now being renamed the Laser 200 for the U.S. market as well. The PP40 printer/plotter uses 4.6" width paper, but prints 26, 40, or 80 characters per line. It prints in black, blue, red, and green on standard roll paper. Since it uses a standard Centronics interface, it can be used with almost any computer, not just the Laser series. Price: about $179.

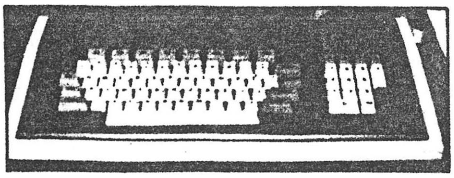
A middle range computer, still in the prototype stage, is the Laser 2001. This 6502-based system is dubbed the Multi-system since, with the appropriate



The Video Technology VZ200 changed its name to the Laser 200.

adapter, it can run ColecoVision or Atari VCS software. It has a 16K ROM with extended Microsoft Basic, 64K of RAM, 256 x 192 graphics resolution, four channels of sound, and interfaces for joysticks, printer, and cassette recorder. No price as yet.

The top of the line computer from V-Tech is the Laser 3000, an Apple work-alike system. Not only does the 3000 run Apple software, but it has many features not in a standard Apple. In particular, it

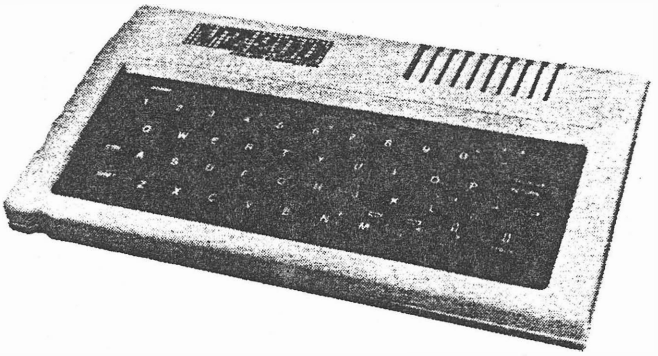


The V-Tech Laser 3000 is an Apple work-alike with many additional features.

has a 24K ROM, 64K of RAM (expandable to 192K), keyboard with 81 keys, eight function keys, built-in 80-column text display, 560 x 192 pixel graphics resolution, four sound channels, built-in Centronics parallel interface, TV sound, and RGB or composite video output. Wow! We can't wait to get one. No price as yet.

p. 202 & 204.

VZ200



The VZ200 is fairly typical of the micros at the bottom end of the price range. It has the now familiar rubber keyboard, although this one seemed a little more positive than some others I have used. Most keys have four functions: the normal mode gives upper-case and digits, holding down the shift key usually results in a graphic character, while the control key adds single key entry of Basic keywords and screen editing facilities (cursor movement, insert and delete). Simultaneous use of the control and return keys while pressing an ordinary key generates function names and the remaining keywords.

The display is not particularly impressive. The picture quality is fine, but there are only 16 lines of 32 characters. In this mode the 'quarter-square' graphics characters give a resolution of 64 x 32, but there is also a graphics mode giving 128 x 64. This makes the VZ200's graphics look limited

when compared with those produced by other systems. The choice of colors is similarly restrictive, with two sets of just four colors for graphics; this does improve to eight colors plus black in character mode.

Other points worth mentioning are the provision of separate TV and video outputs, and the generous lead on the plug-pack power supply unit.

The Basic lacks the sophisticated features of the extended Basic found in the Spectravideo and Tandy Color computers. However, it does include primitive graphics commands (SET, RESET and POINT), as well as a SOUND command which generates a tone with a specified frequency and duration. Another feature that is not always found on cheaper machines is a data file facility (using PRINT# and INPUT#). On the subject of files, I found the VZ200's cassette interface to be much more sensitive to volume levels than other micros that I tested. Once I had found the right setting, things went smoothly.

The documentation was another run-of-the-mill affair. The so-called Basic Reference Manual is actually a tutorial, although there is some reference material at the back. An eight pase User Manual shows how to set up the system, plus a troubleshooting guide. The final part of the package contains listings of 21 simple programs, a few of which might be useful. Even though they appear to be reproductions of actual listings, there is at least one syntax error present. Another snag is that the copyright notice states "No part of this book may be utilised in any form . . . without permission in writing from the Publisher", and nowhere does the publisher state that the programs can be used by a VZ200 owner!

The VZ200 was supplied with a 16k RAM pack which was surprisingly bulky. I found it necessary to keep the TV lead away from the RAM pack to avoid interference. The RAM pack comes complete with a tiny screwdriver to remove the cover fitted to the expansion connector on the back of the VZ200 — a very thoughtful idea. There is also a "peripheral" connector, but the only peripheral mentioned in the manuals is a printer interface. Apparently the VZ200 firmware includes screen dump routines for use with a Seikosha GP100 or GP100A printer.

Programs are provided on cassette: I received some serious software like a cash flow program, but mainly games. Perhaps I had been spoiled by the higher quality graphics of other micros, but I wasn't over impressed with all of the games.

Since the VZ200 is (so far as I know) the cheapest home computer on sale in Australia, my comments could be considered harsh. But providing you are aware of its limitations the VZ200 should be good buy.

Processor	Z80
RAM	8k (Unable to determine usable RAM)
ROM	16k
Ease of use	★ ★
Ease of programming	★ ★ ★
Expansion and accessories	★ ★
Documentation	★
Presentation	★ ★ ★

Benchtested: Vol 4, No 4 (April 1983)

Review machine supplied by Dick Smith Electronics.

APC 4(10) Oct 83
p 77-78.

THE LASER 200

From the land of the Rising Tower Block, Hong Kong, comes a small colour computer called the Laser 200. You may have seen one on show at the Computer Fair: now *Computing Today* has taken delivery of one for our usual in-depth hardware review.

First impressions are of a competitor aimed at the ZX Spectrum, still the leading contender at an end of the market where open warfare is breaking out. Read *Computing Today* next month to find out how it measures up.

Texet TX8000

Price: £98 (soon to be relaunched as Laser for £70)
Use: Home
RAM: 4K
Colour: Yes
CP/M: No
Language: Basic
Interface: Own
Supplier: Texet 061 486 9231

For: Low price, fair Basic as long as graphics aren't important, choice of conventional or one-key programming.
Against: Poor graphics (128 by 48), no sound, below-par internal construction could cause long-term reliability problems.
Conclusion: The same price as a Spectrum, but much less capable. Software is promised for early release, but even the best programmers will find it hard to produce the kind of arcade game so important for the home market. And save £30 by waiting for the Laser.

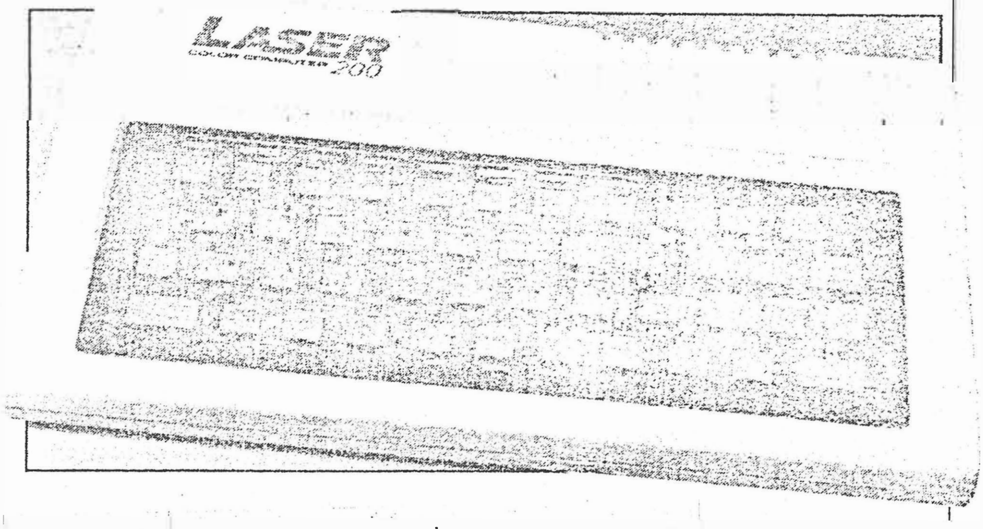
Laser 200

New on the computer scene is the Laser 200 computer, and we'll be taking the lid off it in the **January** issue of *Personal Computing Today*.

Weighing in price wise at £70 for the basic computer console which has a cream plastic case with a brown keyboard made of the same plastic material as the ZX Spectrum. The **Laser 200** is a colour computer but the main drawback is that it only comes with 4K of memory on board. For £30 you can buy a 16K extension RAM pack which plugs neatly into the back of the main console.

Produced in Hong Kong it is being distributed in Britain by Leisure Zone, an associated company of Video Technology and is on sale through retail outlets. According to our street-wise spys the Laser 200 is selling remarkably well.

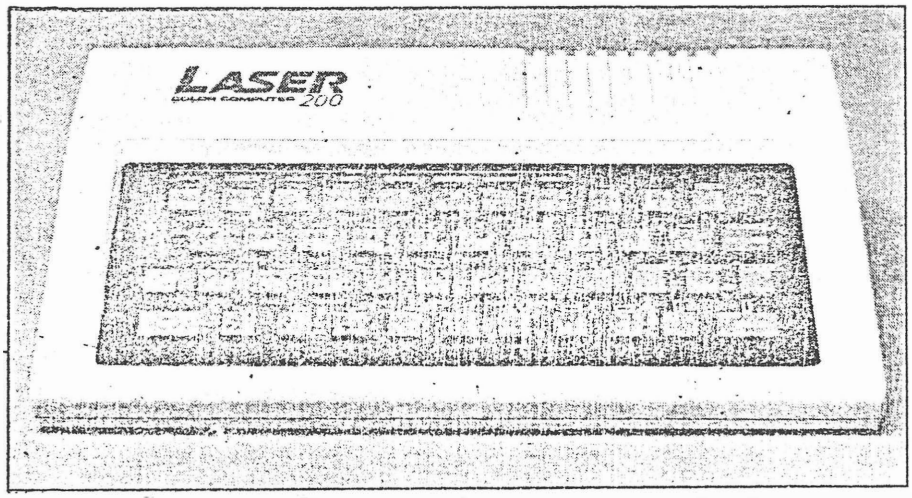
Look out for the January *Personal Computing Today* to find out what the Laser 200's rays are really like.



Peter Green

A LOOK AT THE LASER

Hailing from the sky-scrapered shores of Hong Kong, the Laser 200 is a surprisingly late arrival from this Land of Technology. Has it been worth the wait?



What is this I see before me? Looking rather like a well-fed, albino version of the ZX Spectrum, the Laser 200 is a rather late entry into the low-cost home computer market from Hong Kong. Quite typically, this origin means that it's very cheap indeed — the basic unit retails for £70. However, there is rather more to the story than simply a low price tag, so let's dig a little deeper and see how appealing the Laser is.

A CASE IN POINT

The Laser has been designed along the same general lines as the ZX Spectrum. Covering a slightly larger area than the Spectrum and about twice as thick, it consists of little more than a sloping keyboard with the electronics tucked in underneath. The keys are made of the same hard rubber (or dead flesh, depending on your point of view) as the Spectrum, and number 45 rather than the latter's 40. The case is cream with a dark brown keyboard surround and light brown keys — all the key legends are in white and are easy to read. An LED at the top right of the keys indicates when the computer is powered up.

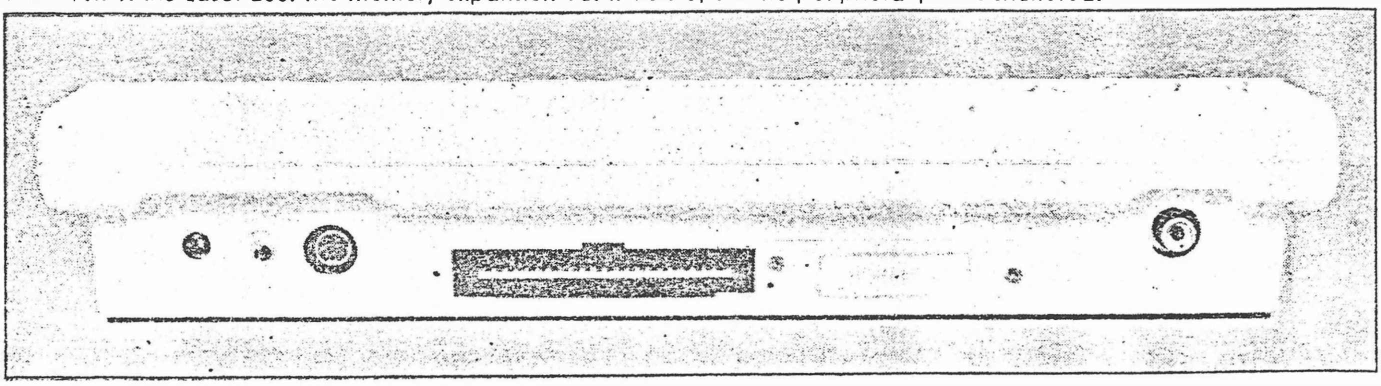
Like the Spectrum, the Laser 200 allows single keystroke entry of BASIC keywords: but unlike the Spectrum it doesn't insist on them. This is good: Beginners will be able to spell the words out in full to begin with, gradually changing over to single key entry as they learn where all the various functions are located. This is easier to pick up than on the Spectrum, where the keyword locations are sometimes a little illogical: on the Laser, words that form natural groupings are located on adjacent keys (like IF-THEN-ELSE, FOR-TO-STEP-NEXT, SET-RESET-POINT and PEEK and POKE). Furthermore all the words in a given grouping need the same type of Shift operation to get the keyword.

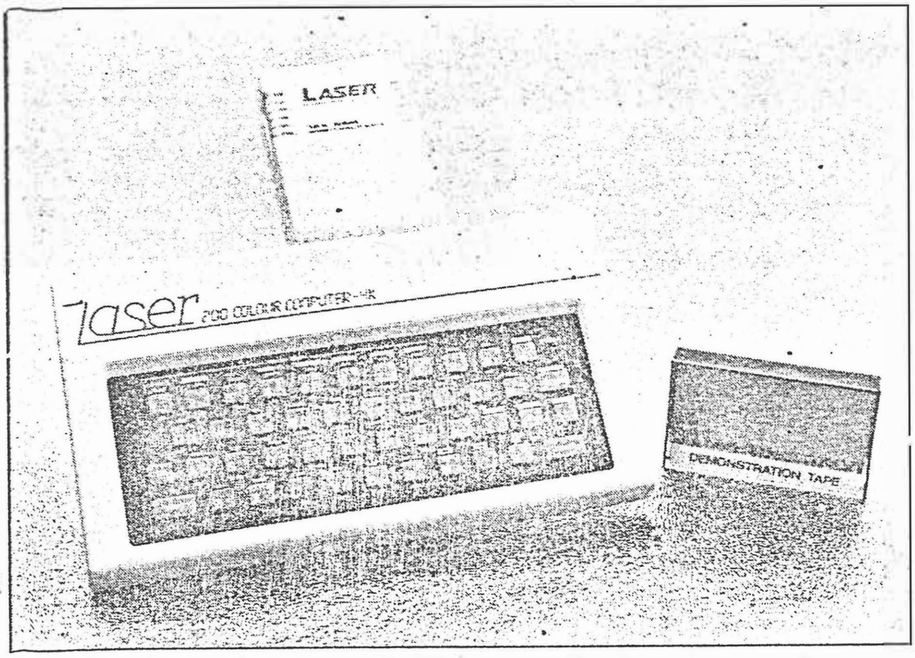
There are two function keys, Shift and Control, and none of the other keys has more than four functions. Unshifted, the keys produce the alphanumeric set and some of the punctuation. Pressing Shift with a key gives the rest of the punctuation, the arithmetic operators and the block graphics. Control and a key gives the BASIC keyword marked above the key, while Control-Return, then Control-key gives the keyword below. (This latter procedure is similar to Sinclair's extended mode). One oddity when using single-key entry; if the keyword requires brackets, as in STR$ (X), then for some functions the leading bracket is printed for you, sometimes it isn't. Oh, well, just remember to keep your eyes on the screen.

On our way round to the back of the computer, we take a slight detour on to the right-hand side where an on/off switch is located. There is, strictly speaking, no real need for this as the Laser isn't mains-powered but uses a separate low voltage power pack like most other computers of this size. However, it's marginally more convenient to flip the switch for a hard reset, should you need one, then reach round and pull out the plug. A trifling point, really.

From left to right across the back panel we have, first of all, the 9 V DC input socket for the power supply, then the tape socket. Yes, socket — singular. Unusually, the Laser has a stereo jack socket rather than the normal twin sockets, but it does have a tape lead supplied with the required connector and the standard

The back of the Laser 200. The memory expansion bus is visible, but the peripheral port is shuttered.





The 16K RAM expansion plugged in. This lies flat rather than sticking up like Sinclair's version.

plugs at the cassette end. No remote control of the cassette recorder motor is provided.

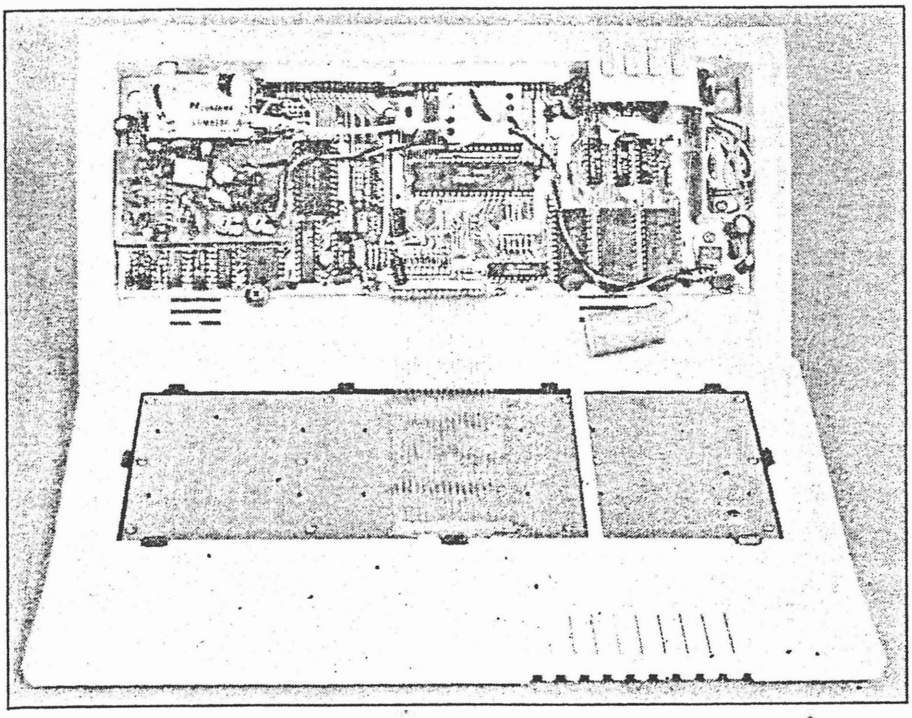
Next comes a monitor output, rare (and commendable) in a machine of this price, followed by the two printed circuit board edge connectors for the memory expansion and peripherals. Finally comes the UHF TV output socket, tuned to Channel 36 or thereabouts as usual.

Thus endeth the guided tour. Also included in the purchase price are a TV lead (too short for comfortable viewing with a domestic TV set, like most other computers), a User manual, a demonstration tape, a BASIC Reference manual and a slim booklet of example programs.

TURNING ON

On power-up the Laser 200 simply says READY. No Microsoft copyright message (for it is they who wrote the BASIC), no message giving the number of bytes free — just READY and a flashing cursor. You can't check how much free memory there is with FRE(0) or SIZE, since such a statement isn't supported. The display is yellow text on a green background, which I promptly messed up by POKEing random graphics all over the place to see what the screen capabilities were.

It isn't terribly tidy inside the Laser, but everything seems to work OK.



This led to an interesting discovery when I tried to clear the screen. There is no key provided for clearing the screen, so it's necessary to use CLS in immediate mode. But with random patterns on the screen the remainder of the line must be cleared with spaces to prevent a syntax error. In doing this, I overshot onto the next line and instead of overwriting that too, the Laser 'opened up' a new line by scrolling the remainder of the screen down a line. An attempt to repeat this on the next line failed, as the cursor refused to move past the end of the second line. The point of all this is that the BASIC is designed to prevent the input of anything longer than two lines, and since the screen is only 32 columns wide, program lines can only be 64 characters long including the line number. This is rather less than the 80-character lines Microsoft normally allows.

Another annoying feature is the action of the Delete key. Instead of being a combined backspace-and-delete, it is necessary to use the cursor keys to position the cursor over the first of the offending characters. Delete then removes that character and pulls the end of the line back by one character, so making a correction could take twice as many keypresses as usual. Fortunately the auto key repeat speeds things up but it was a little difficult to get used to.

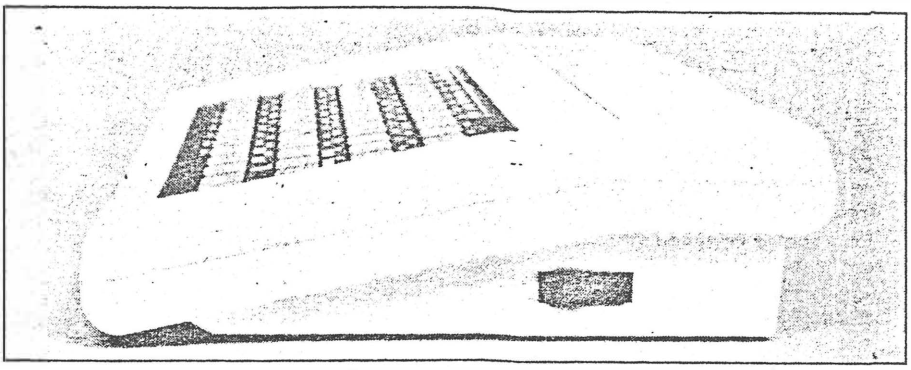
Apart from these quirks the BASIC is pretty much standard Microsoft, with multistatement lines, the usual maths functions, the usual string handling functions (sufficient memory for string operations must be reserved using CLEAR), and the surprising IF-THEN-ELSE which some more expensive machines do not have. Arrays can have up to three dimensions. I/O functions are supported by INP and OUT, and USR calls to machine code routines may be made.

The cassette commands are the standard CLOAD, CSAVE and VERIFY, plus CRUN which loads a program and autoruns it. For some reason the manual insists in quite strong terms that you must always start the tape running before hitting Return during any tape operation: I can understand this for CSAVE, where you might lose some of the header, but not for the other three, and the machine didn't complain when I broke the rules. Named data files may be stored on tape using PRINT#, and loaded into variables using INPUT#.

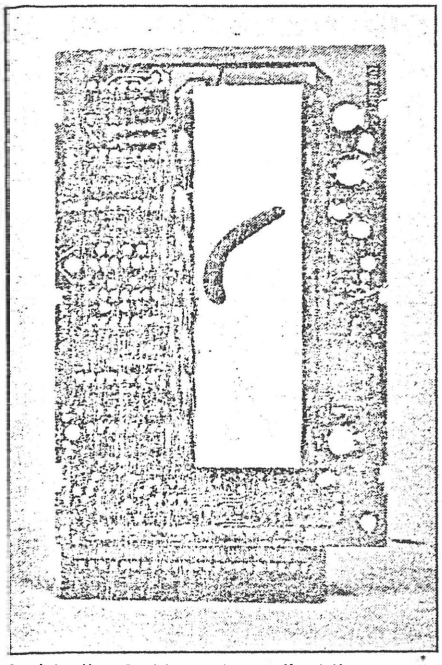
GRAPHICS

There are always two graphics modes. The text mode, which the Laser always defaults to when a program isn't running, is MODE(0) — it

insists on the brackets — and gives a 32 by 16 display. Text is upper case only, with a choice of yellow on green or light brown on dark. Selecting inverse text gives you the same two colour combinations with the foreground and background reversed. Since the Laser uses a separate display code for each of the normal and inverse characters, that takes care of half of the possible 256 displayable characters. The other 128 display codes are assigned to eight repetitions of the 16 text mode block graphics characters, one set for each of the eight foreground colours in this mode (green, yellow, blue, red, buff, cyan, magenta and orange). You can have any background colour you like for the block graphics so long as it's black. Indeed, the only way to get black on the screen at all is as part of a text mode graphics block.



An on-off switch is provided on the side of the Laser.



Inside the RAM pack we find the memory chips under metal shielding, and a switched mode psu down the right of the PCB.

Note the use of display codes rather than ASCII codes: like the PET, Sharp and other machines, to get an 'A' on the screen you can either PRINT CHR$(65) or POKE 28762,1.

In the high-res graphics mode, MODE (1), the pixel resolution is 128 by 64, rather poor by today's standards. The colour set is also restricted in this mode, with a choice of two sets. There's a green background with green, yellow, blue and red foreground colours, or a buff background with buff, cyan, magenta and orange foreground. No text can be displayed in MODE (1), and the only pixel manipulation commands are limited to SET, RESET and POINT (returns the colour of the tested pixel). No line drawing commands, no CIRCLE, no flashing from hardware. Sigh.

Resorting to machine code can give much better possibilities, as in the 'intro' and 'outro' sections of the demo tape. This program is not recommended for epileptics!

SOUND

The SOUND command is not much of an improvement on that of the Spectrum, though it is louder. Two parameters can be specified, to give 31 frequencies and nine different durations. OK for simple tunes and games sound effects, but nothing advanced.

EXPANSION

The 4K user RAM of the basic Laser 200 may be expanded by the addition of a 16K module, which we tested, or a 64K module, which we didn't. The module seemed rather chunky compared to RAM packs for other computers and we couldn't resist opening it up to take a look. Underneath the layers of metal, presumably for RF shielding, we discovered a small switched mode power supply, amongst other things. This is probably generating 12 V and suggests that the price has been kept down by using the older multi-rail supply chips, rather than the modern single rail 5 V versions.

The peripheral port will take an add-on printer interface which will drive the Seikosha GP-100 and GP-100A printers (according to the manual), or any Centronics printer (according to the synopsis on the packaging). The relevant commands are LLIST, LPRINT and COPY; the manual doesn't go into details about what happens to the various colours when the high-res screen is dumped.

Again, according to the packaging there is a light pen and a joystick which may be connected to the peripheral port, though no mention is made of how to program for them. The details are probably included with the accessories, and we were not supplied with either.

The question of possible disc drives is even more vague: the only reference to them is in the list of error messages at the back of the manual, which includes DISK COMMAND as one entry.

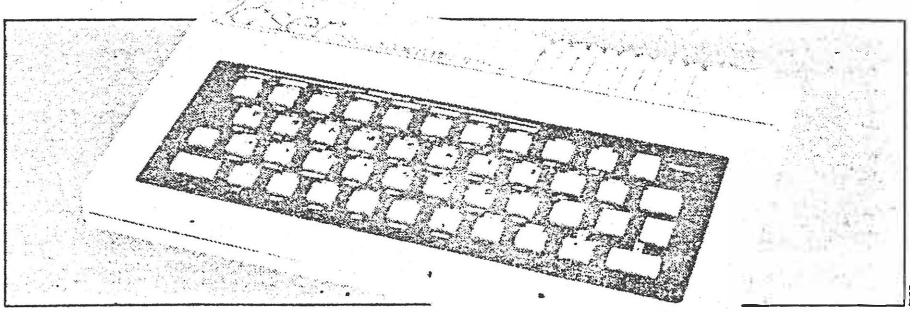
THE DOCUMENTATION

The manuals for the Laser 200 are no worse than those for many other computers, and are better than some. There's the usual smattering of spelling mistakes, most of which are harmless, and the level is pitched at the rank beginner. Unfortunately the manual has equated simplicity with brevity in many places, and a beginner may need rather more explanation of some aspects. The sample programs are all short and quite basic.

I particularly liked page 21, which had a drawing of the screen with SYNTAX ERROR displayed on it, and beneath it the explanation, "This means SYNTAX ERROR . . ."

CONCLUSIONS

Throughout his review I have made comparisons between the Laser 200 and the Spectrum, which one tends to



do instinctively given their similar appearance. In reality this is probably an unfair comparison, because although the Laser costs only £70, the basic computer has only 4K of memory and the price of a 16K RAM pack takes the price up to that of a 16K Spectrum, which offers much better graphics and more facilities for expansion now that the Microdrives and networking are available. (On the other hand, the 64K expansion takes the price to that of a 48K Spectrum). Perhaps a fairer comparison for the basic machine would be one made with the ZX81, another computer intended as a low-cost entry into computing but with an inferior keyboard and no sound and colour.

Unfortunately Sir Clive, with his usual consummate timing in these matters, has just dropped his price to £45 for a ZX81, 16K RAM pack and software cassette, forcing people to decide whether it is worth paying the extra £25 for sound, colour and a quarter of the memory: not to mention the vast amount of software available for the two Sinclair machines which widens the gap even further. It seems that the Laser 200 has fallen between several stools, and it may remain there unless the distributors can stimulate the interest of the commercial software houses.

BENCHMARK	BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8	Average
TIME	1.7	7.0	17.0	17.4	19.3	31.6	48.8	72.5	26.9

FACTSHEET

CPU	Z80A
ROM	16K
RAM	4K (expandable to 16 or 64K)
Language	Microsoft BASIC
Keyboard	45-key multifunction, moving rubber membrane
Display	Text mode: 16 lines of 32 characters, 32 by 64 pixel graphics in eight colours plus black. High-res mode: 64 by 128 pixels in four colours (choice of two sets), no text. TV or monitor output
Cassette	600 baud
I/O	Centronics printer interface, lightpen, joysticks, bus for memory expansion
Sound	Single channel, 31 notes, 9 durations
Costs	Laser 200 £69.95 16K RAMpack £29.95 64K RAMpack £59.95 Printer interface £19.95 Joysticks £19.95 per pair Lightpen £19.95
Supplier	Computers For All, Southfields Industrial Park, 30 Hornsby Square, Laindon, Essex Telephone 0268 418414

p 37-40 4 of 4.

THE LASER – A SH

THE LASER 200 IS THE CHEAPEST COLOUR COMPUTER IN THE UK. BUT WITH A SMALL MEMORY OF ONLY 4K AS STA

If the Laser looks familiar, don't worry — there is a good reason. Those readers with long memories will recall a micro called the Texet TX8000, which we reviewed in our March 1983 issue.

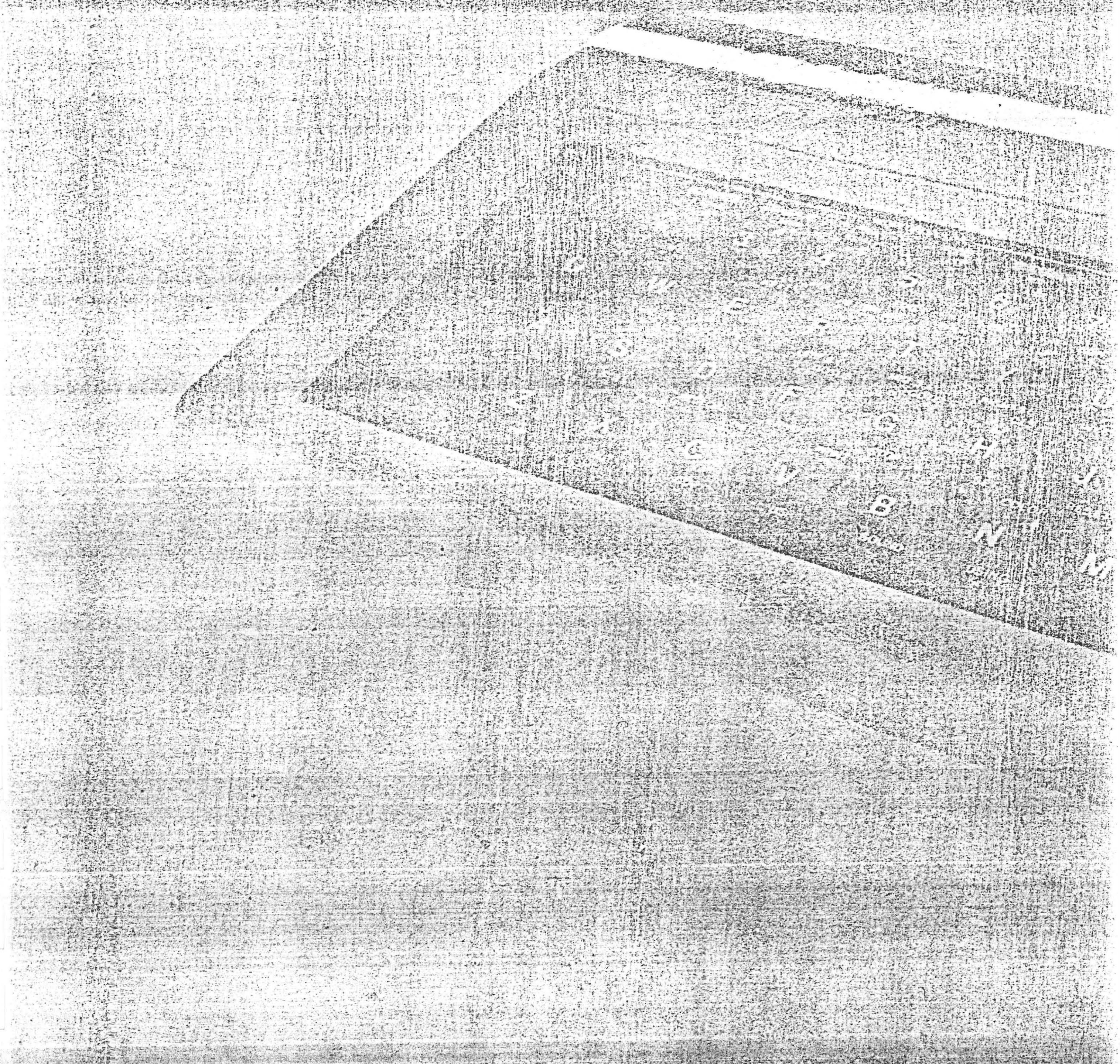
At that time we were not too enthusiastic about the TX8000's future, and this pessimism seems to have been justified. After a couple of months we stopped hearing about it, and it was assumed that the machine was extinct. The recent introduction of the Laser, now being handled in this country by Computers For All, brought the same computer back into circulation in a modified form with £30 chopped off the price. And the Laser may well have a brighter future ahead of it at £70 for the 4K computer plus £30 for a 16K RAM pack.

The machine remains fundamentally similar. It is an elegant-looking micro in a tough plastic case, slightly larger than the Spectrum and about the same size as an Oric. At the heart of the Laser is the familiar Z80A central processor, and a rather inadequate 4K of RAM.

From the outside, the Laser is considerably more impressive than some of the sub-£100 micros we have seen in recent months. Sporting such daring innovations as an on/off switch (one better than the ZX81, Spectrum, Acorn Electron and many others), and a monitor outlet for connection to a monochrome monitor (a major bonus for eye-strained programmers), there is no evidence of over-zealous activity by the cost-cutting department.

Apart from the usual cassette interface and a UHF outlet for connection to a domestic television set, the rear edge of the computer also carries a pair of neat plates. When unscrewed, these uncover a pair of Sinclair-style edge connectors, devoted to the optional 16K RAM pack and a Seikosha dot matrix printer.

The Laser keyboard is rather better



Which Micro Nov 83 p 42 1 of 3

OT IN THE DARK?

OARD; WILL THE NEED FOR AN EXTRA 16K RAM PACK MAKE THE LASER AN EASY TARGET FOR THE £99 SPECTRUM?

than that of some of its rivals, and entering our test programs was notably quick and easy. The light action of the keys, combined with a quiet 'blip' when the appropriate letter is generated, leaves the inexpert user free to look at the keyboard rather than the screen.

Complete non-typists are also catered for, since by depressing the Control key with another key, a complete Basic keyword is produced. Having the option of this method is much more satisfactory than the compulsory usage imposed by the various Sinclair machines, where it often takes longer to find a short keyword on the cluttered keyboard than it would to tap it in, letter by letter.

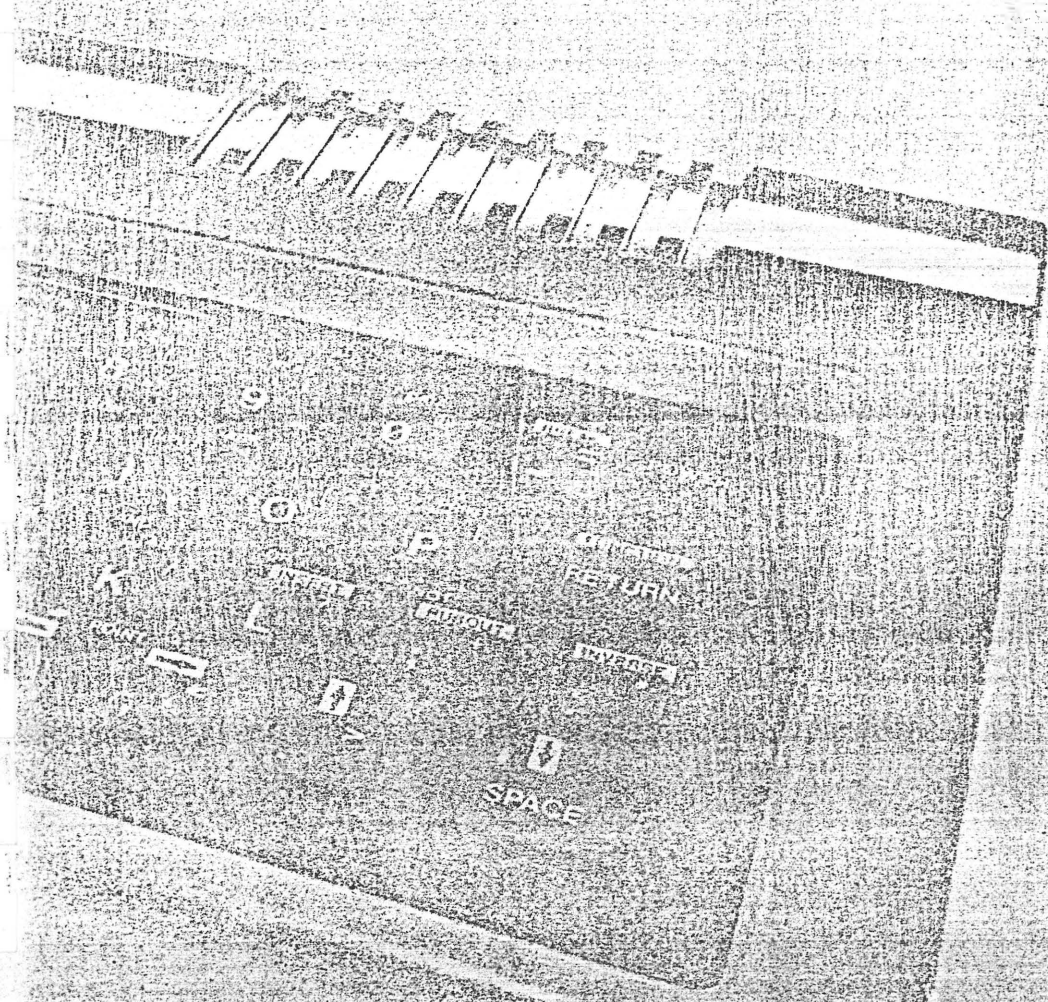
Our only complaint with the keyboard is that when typing at speed, a key would occasionally stick down, and the fast auto repeat would then produce a maddening series of blips as the screen filled with unwanted characters.

Remembering the somewhat crude internal construction of the Texet, we opened up the sturdy case expecting the worst. It was a pleasant surprise to see that the internal layout has been changed, suggesting that Video Technology, the Oriental manufacturer of the Laser, has not been idle in the last few months.

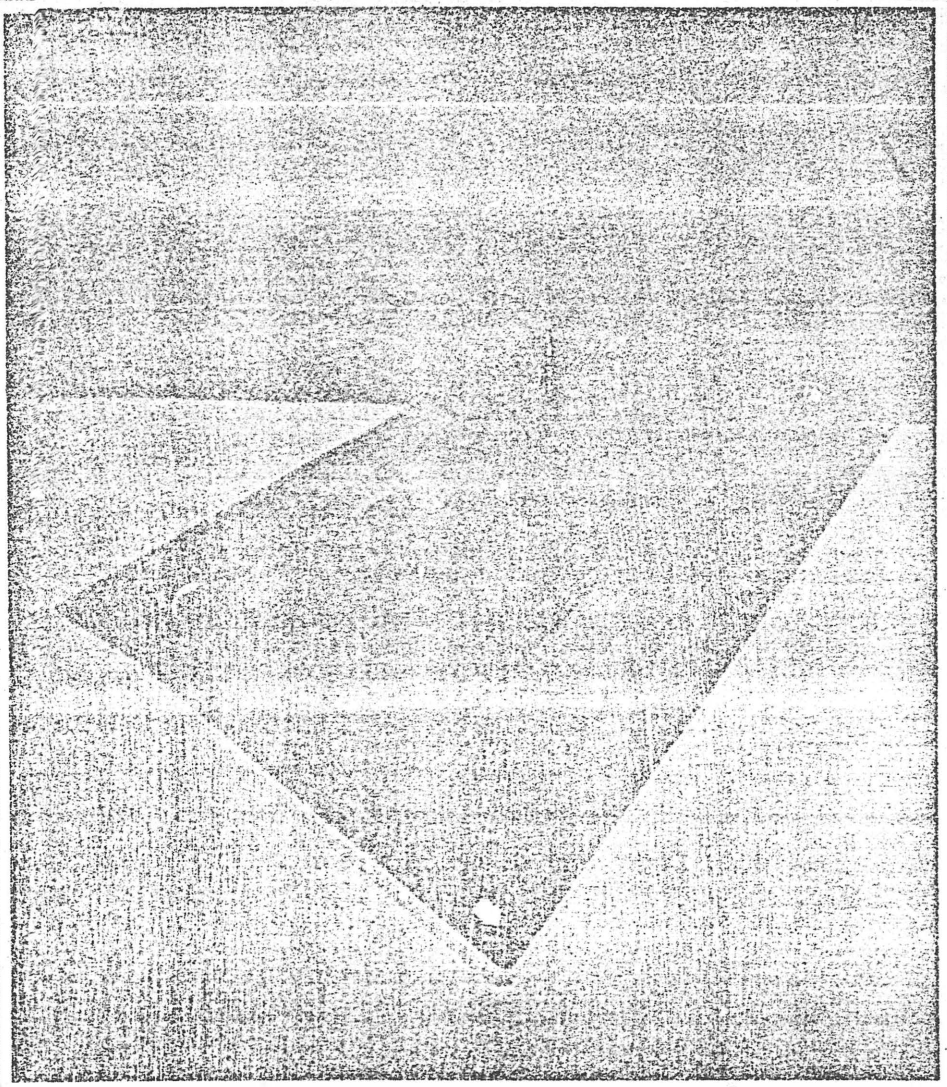
The standard of electronic workmanship is still not as high as that produced by leading European and American manufacturers, but the ugly blobs of glue have vanished and the soldering seems considerably neater than before.

Plugging in the external power supply (now of the type fitted with a separate plug at the end of a lead) we switched on, to see if these changes had been reflected in the Laser's performance as a computer.

The Basic remains very close to the 8K Microsoft which many readers will know. There are still no handy toolkit commands, such as a line or variable trace, line renumber utility or even automatic line numbering, but the Basic is a known



Which Micro Nov 83 p 43 2 of 3.



quantity and not subject to any drastic bugs.

There has been some improvement, however, as we found out when we tried the only thing which caused a complete system crash during our test of the Texet. Previously, typing LLIST or LPRINT (the commands to print or list the program on the printer) when no printer was attached caused the machine to freeze up completely, switching off being the only way of regaining control.

The Laser is sufficiently smart to recognise that there is no printer fitted, and it just ignores printer commands when they are not appropriate.

The quality of the display is good, and the standard green background makes text easy to read. The user is free to change the colours to any combination of the eight available — green, yellow, blue, red, buff, cyan, magenta and orange.

Unlike some recently introduced micros, the Laser has a first-rate screen editor for the alteration of Basic programs. This is very important for beginners who are likely to miss their mistakes until the complete program has been typed in. Correcting the errors is quick and easy with the Laser.

One disappointment is the lack of a lower-case display. In common with a disturbing number of recent releases, the Laser is made to look rather old-fashioned by this omission. Graphics fans may also be rather disappointed by the low resolution available in the so-called 'high resolution' graphics mode — a mere 128 × 64, which means that fine plotting is out.

Graphics plotting is only possible one pixel at a time, using the commands SET and RESET, so graphics displays are likely to involve a lot of laborious work and a lot of program space.

Although one games software house, Abbex, is known to be producing games for the Laser, even the best programmers will find these graphics restrictions a handicap when trying to produce the kind of arcade action which has propelled the Spectrum and its attendant software to the top of the popularity charts.

In the text display mode, a number of chunky graphics characters are available *à la* ZX81, but no user-defined character facility is available.

One bright spot is the provision of a sound facility on the Laser. Producing beeps through an on-board speaker rather than the television set, the resulting sound is similar to that of the Spectrum, though considerably louder.

Sound can be modulated by altering pitch and duration of the note, but there is no provision for white noise in Basic, and in common with the Spectrum, the Laser will need interrupt-driven sound produced by machine code if the action is not to stop while the sound effects take place.

Documentation is quite reasonable for a £70 micro, with a helpful guide to Basic programming enclosed with a brief 'how to switch it on' user guide and a booklet of short Basic program listings. These are mainly short mathematical routines.

As it stands, the Laser is a perfectly competent little machine, but it lacks the sparkle it needs to be a real success.

Recently seen at the Chicago Consumer Electronics Show was the Video Technology Laser 3000, a much more impressive machine with a fast 6502 processor and between 64K and 192K of RAM. With colour graphics capable of a resolution of 560 × 192, it matches all comers for graphics, and the four-channel sound generator is a great advance over the Laser 200's beeping speaker.

This is definitely the way the home computer market is moving, as forthcoming machines like the Atari 600 and Adam show. It's a pity that the 200 has been chosen in favour of the 3000 for Video Technology's entry in the UK market. Let's hope that Computers for All is fast off the mark with the new micro. ■

SPECIFICATIONS

Price: £70 including VAT

RAM: 4K; 16K expansion for £29.95

Keyboard: 45 [illegible] plastic moving key

Processor: Zilog Z80A

Interfaces: UHF, composite video, cassette, printer and memory expansion

Baud Rate: 600

Language: Microsoft Basic

Supplier: Computers for All dealers throughout the UK

FINSBURY BENCHMARK

BENCHMARK TIMINGS (in seconds)

	Laser 200	ZX Spectrum
Test 1	18.0	44.7
Test 2	7.9	8.9
Test 3	18.7	20.7
Test 4	19.8	19.8
Test 5	21.9	23.5
Test 6	35.5	52.6
Test 7	55.0	339.4

COMMENT

The Laser's main claim to fame is that it is the cheapest colour computer available in the UK. Indeed, at its current retail price of £70, it undercuts the main opposition by about £30. But most users will find it necessary to buy the 16K memory expansion, and this raises the price to about the same level as the 16K Oric or Spectrum.

The 4K may be just about adequate for a few experiments in the art of programming, but we would be very surprised if many games for the unexpanded machine become available.

But at the £100 mark, the 16K Spectrum offers real headaches for any potential rival with its unmatched software support and a good choice of add-on peripherals.

3 of 3.

in and out of sync

David H. Ahl
David Grosjean

What's a Brand X doing in *SYNC* Magazine?

With improving technology and intensifying competition in the small computer market, more and more computers are available at prices within a few steps of the Timex/Sinclair units. Our sister publication, *Creative Computing,* evaluates many of these systems. We would like to share these reviews with those of you considering another computer.

In addition, we will sometimes take a program or two and show what it would be like to write and run the program on the Brand X computer compared to the Timex/Sinclair. You will probably find these tutorials a useful aid for converting programs from other sources to your Timex/Sinclair computer.

The Video Technology VZ200

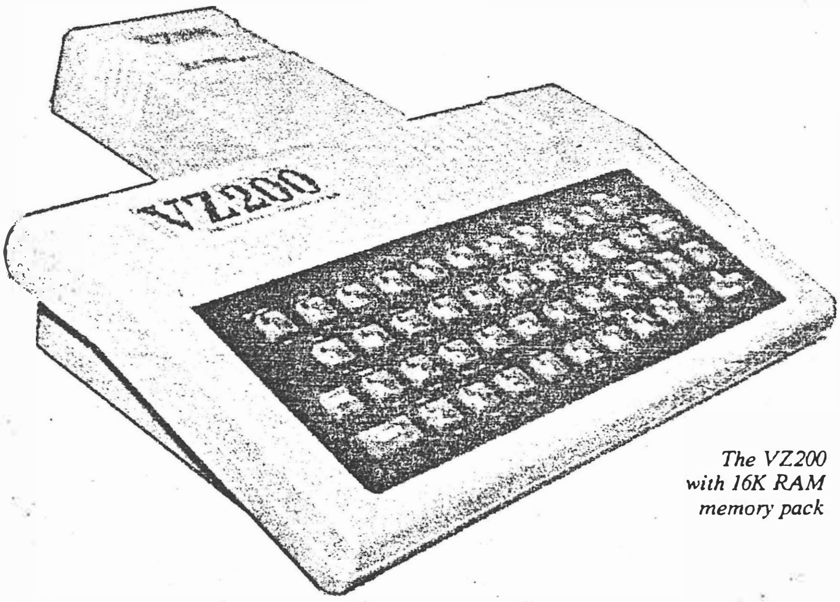
David H. Ahl

The Video Technology VZ200 is a compact microcomputer with a great deal of capability and many unexpected features at a very attractive price. Z80

The VZ200 is based on the 6502 microprocessor (used in the Apple, Commodore, and Atari computers). It comes with a 12K ROM and a sparse 4K RAM. The ROM includes the monitor and an excellent implementation of Microsoft Basic. The RAM can be expanded with either a 16K or 64K module.

The computer is 11.4" x 6.3" x 2". Two-thirds of the top is taken up by the keyboard. The 45 keys are "Chiclet" style rubber with a very short throw. Touch typing is possible only in a rather limited way. Although the key spacing is the same as on a regular typewriter, the feel is different. Much more disasterous for touch typing is the use of a single shift key and a space key instead of a space bar. Several keys do not have the expected characters; e.g., the question mark is on the L key.

On the brighter side, each key provides several functions in addition to typing a character. All the Basic commands, keywords, and functions can be produced by holding the control key (or control and RETURN) while the key is pressed. Each key produces two Basic keywords and one or two regular characters. This is most welcome since on the computers which use a single keystroke the number of Basic keywords is limited to the number of keys.



The VZ200 with 16K RAM memory pack

When a key is pressed, a short "beep" indicates one keystroke. If the key is held down, it automatically repeats with a beep indicating each key entry.

The computer has an on/off light on top and an on/off switch on the side.

The Basic Language

The Basic includes 9 commands, 27 statements, 11 arithmetic functions, 9 string functions, 7 graphics and sound functions, and the expected arithmetic, relational, and Boolean operators.

Among the statements that we do not always see in a computer in this price range are: INP (reads the contents of input ports); OUT (sends values to output ports); USR (calls an assembly language subroutine); and COPY (copies the content of the screen to a printer).

We were also pleased to find both PRINT USING and PRINT @ implemented. The latter is useful for printing at different screen locations without having to use blank print lines or tabs. However, a tab function is also available.

On-Screen Editing

Full on-screen editing makes it a pleasure to program on the VZ200. The line to be

1 of 2. (p. 17-22.)

For editing, the directional keys put the cursor wherever you want it on the screen.

edited is listed, by itself, with the whole program or with a group of lines. The cursor is moved by the directional keys to the character to be changed. Type the change, move the cursor to the end of the line, and type RETURN. Voila! The change is made. On-screen editing can also use DELETE, INSERT, and RUBOUT.

We had two small problems with on-screen editing. First, it was all too easy to hit the shift key instead of the control key because the cursor directional keys are activated by pressing the control key on the left and a directional key on the right. Probably the user can adapt to this after some practice. Second, after a while the editing buffer seemed to overflow and further editing was not accepted. Admittedly, we were trying to push the computer over the brink, so it is unlikely that this will be a problem in normal use.

Video Display

The VZ200 produces a composite video signal for a monitor and an RF signal on Channel 2 or 3. We found the monitor signal rock steady, whereas the RF signal required very precise fine tuning.

Output is in one of two modes: low-resolution text and graphics or medium-resolution graphics only. In the mixed mode, the display has 16 lines of 32 characters each. Alphabetic characters are available in uppercase only. Graphics are made from 16 characters which divide each screen location into four boxes with all combinations as on the ZX/TS computers.

Each of these characters can be turned on in any of eight colors. The off portion shows as black which can be considered a ninth color. Alphanumerics are displayed either as yellow on green or yellow on buff. Individual characters or the entire screen can be changed to inverse. Only one background color, green or buff, can be used at a time, and it does not affect the color of the graphics characters.

Low-resolution graphics characters can be typed into programs directly from the keyboard or called with CHR$(128) to CHR$(255) from a program.

In medium-resolution graphics mode, the screen is 128 x 64 pixels. Each pixel is turned on by the command SET (x,y) and turned off by RESET (x,y); POINT (x,y) examines whether a pixel is on or off. The first two commands are equivalent to PSET and PRESET in some other computers.

In this graphics mode, only three colors plus the background color are available simultaneously.

Any RAM location, including screen locations, can also be changed and examined by POKE and PEEK.

Musical Sounds

The single sound channel can produce 31 frequencies (2 1/2 octaves) and nine note durations (from a dotted half note to a thirty-second note). The command takes the structure: SOUND (p,d) where p is the pitch (1 to 31; 0 for a rest) and d is the duration.

Problems

In pushing the computer to the brink, we found several situations in which the only way of recovery was to turn the computer off. Even BREAK (the equivalent of RESET on some other machines) failed to return control to the user.

The most common irrevocable condition was LLIST which normally lists a program

on the line printer. However, if no printer is attached, the computer hangs. This is particularly bad because the rubberized keys tend to bounce a bit, and it is easy to type LLIST instead of just plain LIST. If you have a long program in the computer and have to turn it off because it hangs up, as we did four or five times, you are forgiven if you become a bit surly toward the machine. The surest cure is to use Control/4 to list a program. After a while we learned to do this.

Other things that would hang the machine are in the same family, i.e., trying to use a peripheral device that is not attached. In some cases the VZ200 gave an error message, but in others it went into never-never land.

We also had a problem loading the programs from the demo tape. We tried three recorders, including a high quality digital unit, but all the VZ200 would say was "FOUND T: Program Name." Since we saw the programs load at CES, we assume we got a faulty demo tape.

Peripherals

The interface to a standard cassette recorder operates at a Baud rate of 600 bps. Although this is somewhat slower than other new computers which have rates up to 2400 bps, nevertheless it is twice as fast as machines of just a few years ago. A program that fills the entire 4K of memory loads in about 54 seconds; a 16K program loads in about four minutes. Bear in mind, however, that most 16K programs do not use 16K of code because much of the RAM is taken by dimensioned arrays and the like.

The manufacturer specifications note that a peripheral expansion bus is built-in; however, we are not quite sure what this means. It appears that expansion modules, presumably, to be connected to printers, modems, or other external devices, can be plugged into the back of the computer.

The V-Tech printer is a Seikosha unit which we have previously found to be satisfactory and cost effective. It requires an interface module which plugs into the interface bus. Since the Seikosha printer uses a standard Centronics parallel signal, presumably other printers with similar signal requirements could be used, although they will probably not reproduce the screen graphics correctly.

Documentation

Included with the VZ200 are a 149-page Basic Reference Manual, a 24-page booklet of 21 Basic Application Programs, and an eight-page User Manual describing how to set up the system.

While some of the documentation obviously shows its Chinese (Hong Kong) heritage, the majority is well written, if not awfully well edited. The Basic manual provides a good introduction to the rudiments of the language although some of the sample programs leave something to be desired (the one to illustrate arrays is particularly bad). POKE and PEEK are explained in only the most cursory way, and we have no idea what the "New Characters Code" chart on p. 104 is for. Also, sadly lacking is an index which is very useful in a reference manual.

On the other hand, the manual is as good as most and better than many. It is just a shame that documentation is the weak spot of so many otherwise excellent computers.

Summary

All in all, the Video Technology folks in Hong Kong have done an excellent job producing a versatile small computer. We were impressed with the excellent implementation of Microsoft Basic, full on-screen editing, repeat keys, and easy-to-use graphics features. The idiosyncrasies were a bit annoying, but owners will get used to them and probably not notice them after a week or two of use. Bottom line: the VZ200 is a great value for the suggested price of under $100.

Video Technology (U.S.), Inc., 2633 Greenleaf, Elk Grove Village, IL 60007.

2 of 2. p. 17-22.

Nov/Dec Sync 1983.

(from 2 of 2)

ill probably use only a fraction of the y rection potential. Resolution with any lor pen is 0.2mm, and drawing speed 52 mm per second.

The graphics commands recognized the PP40 are nearly as rich and var- as those on much larger and more pensive plotters. The PP40 can pro-ice 15 different types of dotted lines, as ell as a solid line. It can also produce ordinate axes automatically.

The draw command (D) draws a line tween any number of x,y point pairs, hile relative draw (J) draws a line from e present location to an x,y point pair. ove and relative move function simi-rly, but with the pen up.

The color command (C) selects a pen lor, scale set selects one of 64 charac-r sizes, and alpha rotate selects one four directions for the printing of phanumeric characters.

The CC40 has three initialization mmands: A initializes everything and its the plotter in text mode; I causes e present pen position to be taken as e starting point; and H moves the pen the home position with the pen up.

The only bone we have to pick is that e plotter requires that commands and parators (commas) be sent to the plot- enclosed in quotation marks in an RINT statement. Most other modern otters do not require quotes. For ample, a draw command between ree point pairs must be sent to the 240 as:

```
80 LPRINT "D"; X1;",";Y1;", ";
2;", ";Y2;",0,0"
```

n other plotters, this line would read:

```
80 LPRINT "D" X1,Y1 X2,Y2 0,0
```

As might be expected, the PP40 does t draw true diagonal lines. Instead, ese lines are produced as a series of rizontal or vertical straight lines with nall steps to create the diagonal direc-on. These steps are evident in the spiral ot shown in Figure 9.

ocumentation

The user manual for the PP40 is bet-r than many of the manuals that come ith many other Hong Kong products, t it is still nothing to brag about. All e graphics commands are described in condensed half-page table. Fortu-tely, the second half of the 38-page anual is devoted to six example plots. ogram listings are provided for three mputers: Laser/V-Tech 200 (standard icrosoft Basic), Apple II (Applesoft sic), and Dragon 32 (same as Radio ack Color Computer). By studying ese programs, you should be able to determine how each text and graphics command functions.

The Bottom Line

Frankly, we like the PP40. It is not a professional, full-function plotter, nor does it take the place of a full-size printer. However, as an inexpensive output device that can do both printing and plotting, it does an admirable job.

The graphics command structure is somewhat cumbersome; diagonal lines are not truly straight; and the documentation could be improved upon. Nevertheless, these are small inconveniences against the good performance, compact size, and low ($199) cost of the PP40.

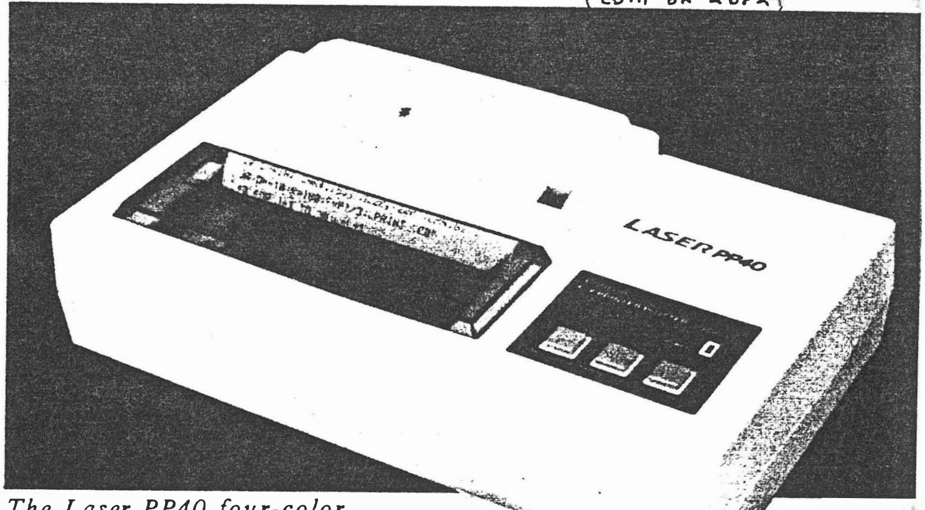
For more information, contact Video Technology, 2633 Greenleaf Ave., Elk Grove Village, IL 60007. (312) 640-1776. END

Laser PP40 Printer/Plotter

The Laser PP40 is an inexpensive ($199) four-color printer/plotter from Video Technology. It has a Centronics parallel interface so it is suitable for use with a wide range of computers, not just the machines from Video Technology. It uses 4 ½" wide roll paper, so it is not suitable for business correspondence; however, for low-cost plotting it is an excellent unit.

The PP40 is one of the smallest printer/plotters we have seen, measuring a diminutive 9.5" x 4.5" x 2.1". An external 8-volt, 1500 ma power supply is also furnished. On the outside of the case we find a rocker off/on switch, red

(cont on 2 of 2)



The Laser PP40 four-color printer/plotter.

```
  !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
@  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _
`  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
p  q  r  s  t  u  v  w  x  y  z  {  ¦  }  ~  ⊠
```

Figure 6. Character set of Laser PP40 in size 1.

```
!"#$%&'()*+,-./0123456789
:;<=>?@ABCDEFGHIJKLMNOPQRS
TUVWXYZ[\]^_`abcdefghijklm
nopqrstuvwxyz{¦}~⊠

10 LPRINT "Character Set"
20 LPRINT CHR$(18);"S2":LPRINT CHR$(17)
30 FOR I=32 TO 127
40 LPRINT CHR$(I);
50 NEXT
60 LPRINT:LPRINT CHR$(18);"S1,C0,A"
```

Figure 7. Character set in size 2 and program used to produce it.

LED power indicator, and three press switches for paper feed, pen change, and color change. On the back are connectors for the power input and Centronics-type interface cable.

To connect the PP40, you will need a cable from your computer with a Centronics-type connector. Some computers such as the Laser 200, Vic-20, TI 99/4A, and Timex/Sinclair 1000 require a separate interface, while on higher-end units this interface is built in.

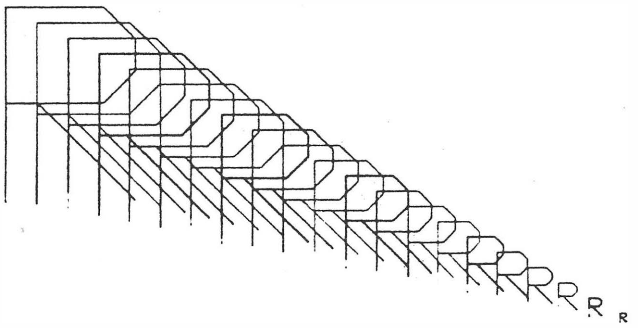
Paper loading is very simple, as are pen mounting and pen changing. The PP40 comes with one roll of paper and four pens with fine ball tips (black, red, green, and blue). Additional paper rolls are available from office supply dealers, while replacement pens must be purchased directly from V-Tech. Although it is not mentioned in the manual, we suggest removing the pens from the unit and replacing their covers if you plan to let the PP40 stand idle for more than a day or so.

On the bottom of the unit is a small plate that covers a DIP switch. One switch selects whether carriage return implies line feed or not, and the other selects 40- or 80-column printing (spelled on the box, "coloum"). Forty-column printing produces 11 characters per inch and 5.5 lines per inch. Eighty-column printing uses a much smaller character size, and produces twice the vertical and horizontal density (22 cpi and 11 lpi). See Figure 9. Using this character size (0), the print speed is 10 cps; the larger the character, the slower the print speed.

The PP40 has a character set of 95 ASCII characters (see Figure 6). In the 40-column printing mode, characters are produced in size 1. In the graphics mode, the PP40 can produce 64 character sizes; the second size is shown in Figure 7, and sizes 0 to 20 are shown in Figure 8. Size 63 is very large indeed with each letter measuring 2" x 3".

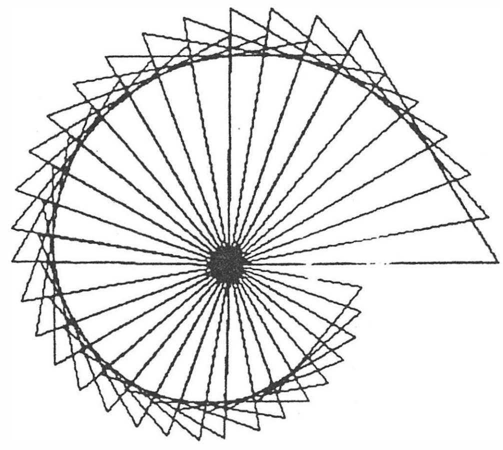
Graphics Mode

In the graphics mode, the PP40 can produce plots 96mm (3.7") wide in the x direction by 6.55 meters (over 21 feet!) long in the y direction. The x direction is divided into 480 steps each 0.2mm in size; the y direction can have up to 32,768 steps. In reality, however, you



```
10 LPRINT "Different Character Sizes"
20 LPRINT CHR$(18);"R0,-200"
30 LPRINT "I"
40 FOR I=0 TO 20:LPRINT "HR20,-3"
50 LPRINT "IC";I;",S";20-I;",PR"
60 NEXT
70 LPRINT:LPRINT "S1,C0,M0,-20":LPRINT"A

70 LPRINT:LPRINT"S1,C0,M0,-20":LPRINT"A"
```

Figure 8. The letter R in the first 21 out of 64 character sizes, and the program to produce the plot.



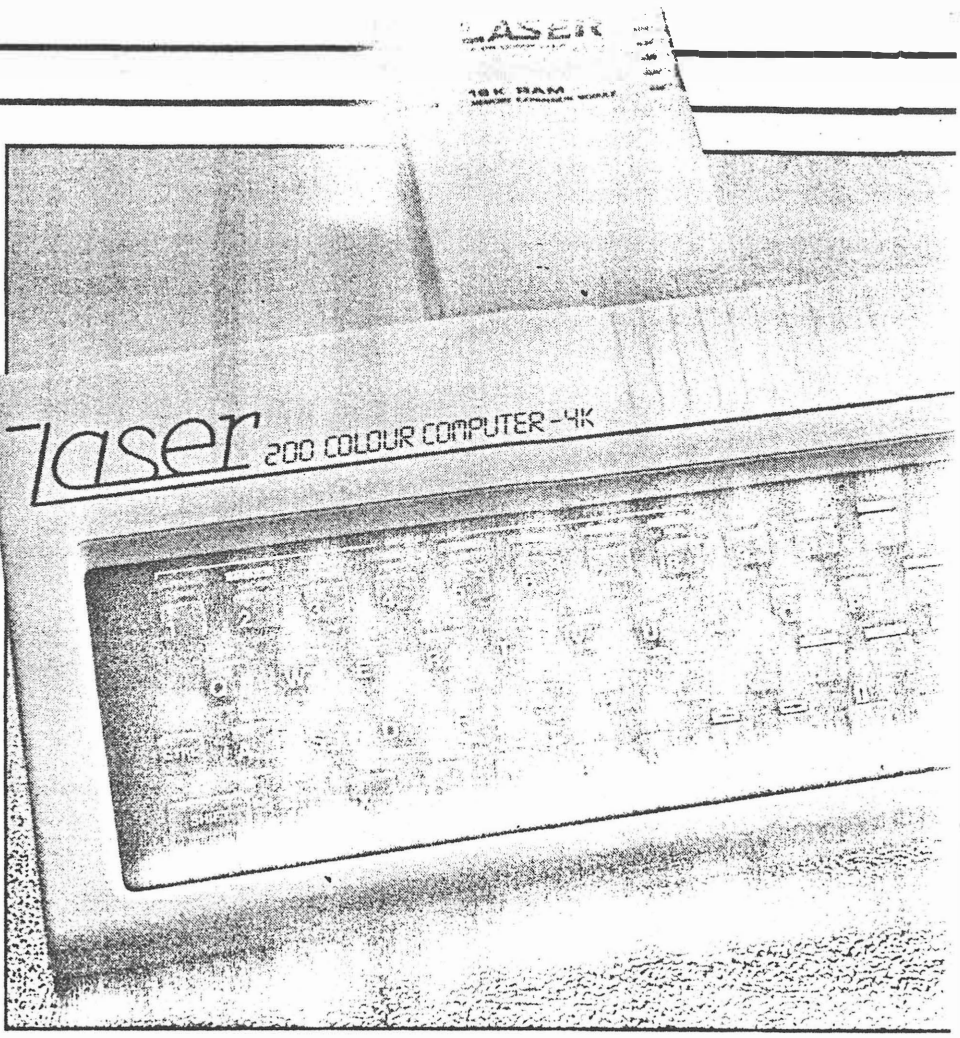
```
10 LPRINT "Spiral Pattern":PI=3.14159
20 LPRINT CHR$(18);"M220,-200":LPRINT"I"
30 D=-10:R=180:F=PI/3:LPRINT "C3"
40 FOR J=1 TO 30
50 D=D+10
60 R=R-3:K=D*PI/180:Y1=R*SIN(K):X1=R*COS(K)
70 Y2=R*SIN(K+F):X2=R*COS(K+F)
80 LPRINT "D";X1;",";Y1;",";X2;",";Y2;",0,0"
90 NEXT J
100 LPRINT "M-300,-150":LPRINT "C0,A"
```

Figure 9. A spiral of triangles of decreasing size. The program listing was produced in 80-character text mode with character size 0.

2 of 2.

Peter Green

Hailing from the sky-scrapered shores of Hong Kong, the Laser 200 is a surprisingly late arrival from this Land of Technology. Has it been worth the wait?



LASER

What is this I see before me? Looking rather like a well-fed, albino version of the ZX Spectrum, the Laser 200 is a rather late entry into the low-cost home computer market from Hong Kong. Quite typically, this origin means that it's very cheap indeed — the basic unit retails for £70. However, there is rather more to the story than simply a low price tag, so let's dig a little deeper and see how appealing the Laser is.

A case in point

The Laser has been designed along the same general lines as the ZX Spectrum. Covering a slightly larger area than the Spectrum and about twice as thick, it consists of little more than a sloping keyboard with the electronics tucked in underneath. The keys are made of the same hard rubber (or dead flesh, depending on your point of view) as the Spectrum, and number 45 rather than the latter's 40. The case is cream with a dark brown keyboard surround and light brown keys — all the key legends are in white and are easy to read. An LED at the top right of the keys indicates when the computer is powered up.

Like the Spectrum, the Laser 200 allows single keystroke entry of BASIC keywords: but unlike the Spectrum it doesn't insist on them. This is good. Beginners will be able to spell the words out in full to begin with, gradually changing over to the single key entry as they learn where all the various functions are located. This is easier to pick up than on the Spectrum, where the keyword locations are sometimes a little illogical: on the Laser, words that form natural groupings are located on adjacent keys (like IF-THEN-ELSE, FOR-TO-STEP-NEXT, SET-RESET-POINT and PEEK and POKE). Furthermore all the words in a given grouping need the same type of Shift operation to get the keyword.

There are two function keys, Shift and Control, and none of the other keys has more than four functions. Unshifted, the keys produce the alphanumeric set and some of the punctuation. Pressing Shift with a key gives the rest of the punctuation, the arithmetic operators and the block graphics. Control and a key gives the BASIC keyword marked above the key, while Control-Return, then a key gives the keyword below. (This latter procedure is similar to Sinclair's extended mode). One oddity when using single-key entry; if the keyword requires brackets as in STR$ (X), then for some functions the leading bracket is printed for you, sometimes it isn't. Oh, well, just remember to keep your eyes on the screen. . . .

On our way round to the back of the computer, we take a slight detour on to the right-hand-side where an on/off switch is located. There is, strictly speaking, no real need for this as the Laser isn't mains-powered but uses a separate low voltage power pack like most other computers of this size. However, it's marginally more convenient to flip the switch for a hard reset, should you need one, than reach round and pull out the plug. A trifling point, really.

From left to right across the back panel we have, first of all, the 9 V DC input socket for the power supply, then the tape socket. Yes, socket — singular. Unusually, the Laser has a stereo jack socket rather than the normal twin sockets, but it does have a tape lead supplied with the required connector and the standard plugs at the cassette end. No remote control of the cassette recorder motor is provided.

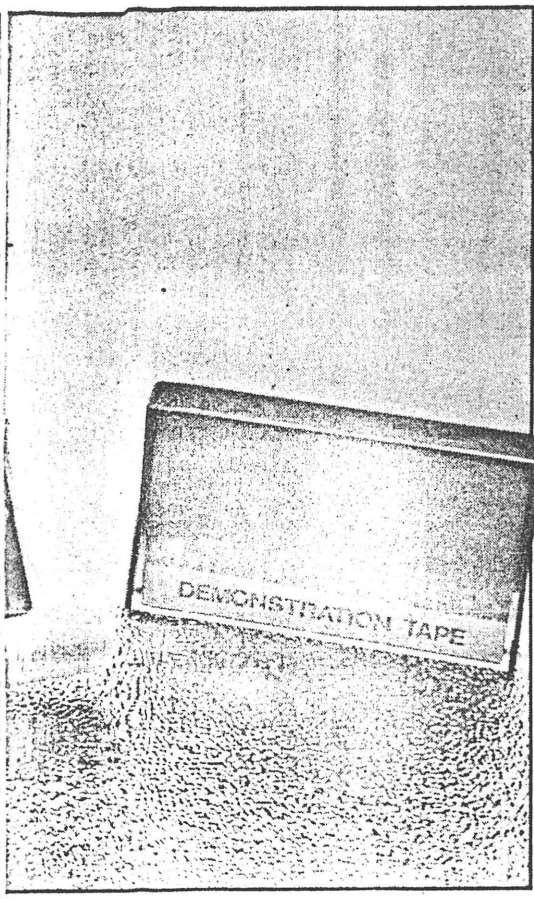
Next comes a monitor output, rare (and commendable) in a machine of this price, followed by the two printed circuit board edge connectors for the memory expansion and peripherals. Finally comes the UHF TV output socket, tuned to Channel 36 or thereabouts as usual.

Thus endeth the guided tour. Also included in the purchase price are a TV lead (too short for comfortable viewing with a domestic TV set, like most other computers), a User manual, a demonstration tape, a BASIC Reference manual and a slim booklet of example programs.

Turning on

On power-up the Laser 200 simply says READY. No Microsoft copyright message (for it is they who wrote the BASIC), no message giving the number of bytes free — just READY and a flashing cursor. You can't check how much free memory there is with FRE(0) or SIZE, since such a statement isn't supported. The display is yellow text on a green background, which I promptly messed up by POKEing random graphics all over the place to see what the screen capabilities were. This led to an interesting discovery when I tried to clear the screen. There is no key provided for clearing the screen, so it's necessary to use CLS in immediate mode. But with random patterns on the screen the remainder of the line must be cleared with spaces to prevent a syntax error. In doing this, I overshot onto the next line and instead of overwriting

1 of 3 Series



200

that too, the Laser 'opened up' a new line by scrolling the remainder of the screen down a line. An attempt to repeat this on the next line failed, as the cursor refused to move past the end of the second line. The point of all this is that the BASIC is designed to prevent the input of anything longer then two lines, and since the screen is only 32 columns wide, program lines can only be 64 characters long including the line number. This is rather less than the 80-character lines Microsoft normally allows.

Another annoying feature is the action of the Delete key. Instead of being a combined backspace-and-delete, it is necessary to use the cursor keys to position the cursor over the first of the offending characters. Delete then removes that character and pulls the end of the line back by one character, so making a correction could take twice as many keypresses as usual. Fortunately the auto key repeat speeds things up but it was a little difficult to get used to.

Apart from these quirks the BASIC is pretty much standard Microsoft, with multistatement lines, the usual maths functions, the usual string handling functions (sufficient memory for string operations must be reserved using CLEAR), and the surprising IF-THEN-ELSE which some more expensive machines do not have. Arrays can have up to three dimensions. I/O functions are supported by INP and OUT, and USR calls to machine code routines may be made.

The cassette commands are the standard CLOAD, CSAVE and VERIFY, plus CRUN which loads a program and autoruns it. For some reason the manual insists in quite strong terms that you must always start the tape running before hitting Return during any tape operation: I can understand this for C SAVE, where you might lose some of the leader, but not for the other three, and the machine didn't complain when I broke the rules. Named data files may be stored on tape using PRINT #, and loaded into variables using INPUT #.

Graphics

There are two graphics modes. The text mode, which the Laser always defaults to when a program isn't running, is MODE(0) — it insists on the brackets — and gives a 32 by 16 display. Text is upper case only. Selecting inverse text gives you the same two colour combinations with the foreground and background reversed. Since the Laser uses a separate display code for each of the normal and inverse characters, that takes care of half of the possible 256 displayable characters. The other 128 display codes are assigned to eight repetitions of the 16 text mode block graphics characters, one set for each of the eight foreground colours in this mode (green, yellow, blue, red, buff, cyan, magenta and orange). You can have any background colour you like for the block graphics so long as it's black. Indeed, the only way to get black on the screen at all is as part of a text mode graphics block.

Note the use of display codes rather than ASCII codes: like the PET, Sharp and other machines, to get an 'A' on the screen you can either PRINT CHR$(65) or POKE 28762,1.

In the high-res graphics mode, MODE (1), the pixel resolution is 128 by 64, rather poor by today's standards. The colour set is also restricted in this mode, with a choice of two sets. There's a green background with green, yellow, blue and red foreground colours, or a buff background with buff, cyan, magenta and orange foreground. No text can be displayed in MODE (1), and the only pixel manipulation commands are limited to SET, RESET and POINT (returns the colour of the tested pixel). No line drawing commands, no CIRCLE, no flashing from hardware. Sigh.

Resorting to machine code can give much better possibilities, as in the 'intro' and 'outro' sections of the demo tape. This program is not recommended for epileptics!

Sound

The SOUND command is not much of an improvement on that of the Spectrum, though it is louder. Two parameters can be specified, to give 31 frequencies and nine different durations. OK for simple tunes and games sound effects, but nothing advanced.

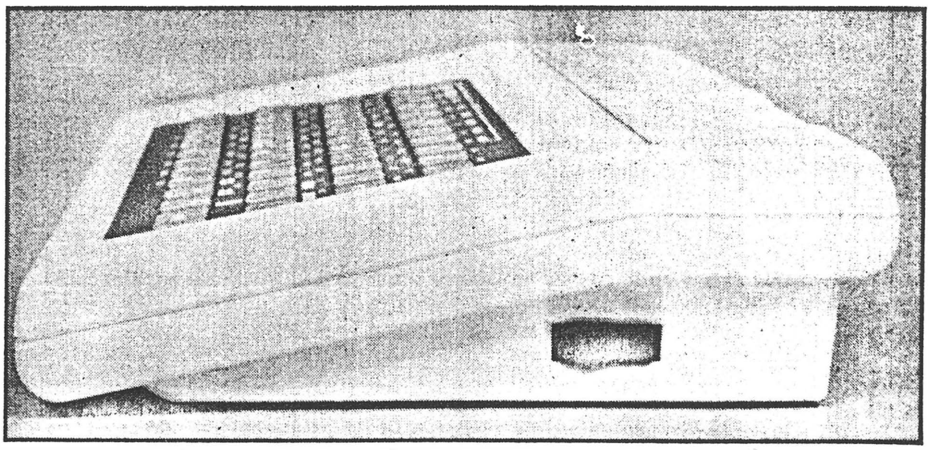
Expansion

The 4K user RAM of the basic Laser 200 may be expanded by the addition of a 16K module, which we tested, or a 64K module, which we didn't. The module seemed rather chunky compared to RAM packs for other computers and we couldn't resist opening it up to take a look. Underneath the layers of metal, presumably for RF shielding, we discovered a small switched mode power supply, amongst other things. This is probably generating 12 V and suggests that the price has been kept down by using the older multi-rail supply chips, rather than the modern single rail 5 V versions.

The peripheral port will take an add-on printer interface which will drive the Seikosha GP-100 and GP-100A printers (according to the manual), or any Centronics printer (according to the synopsis on the packaging). The relevant commands are LLIST, LPRINT and COPY; the manual doesn't go into details about what happens to the various colours when the high-res screen is dumped.

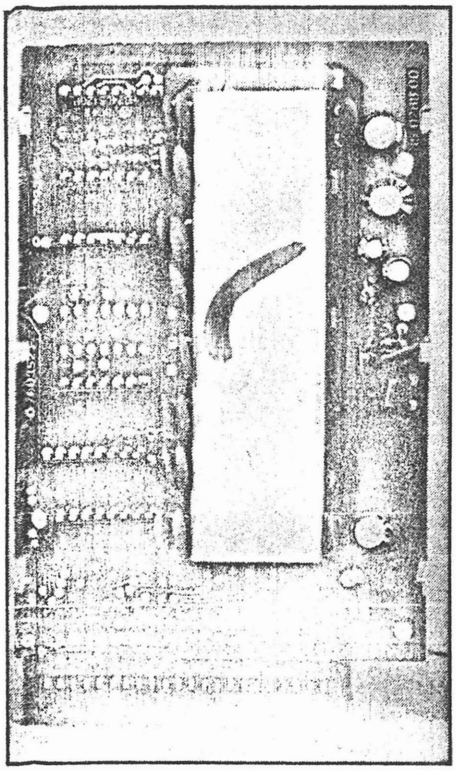
Again, according to the packaging there is a light pen and a joystick which may be connected to the peripheral port, though no mention is made of how to program for them. The details are probably included with the accessories, and we were not supplied with either.

The question of possible disc drives is even more vague: the only reference to them is in the list of error messages at the back of the manual, which includes DISK COMMAND as one entry.



An on-off switch is provided on the side of the Laser.

2 of 3



Inside the Ram pack we find the memory chips under metal shielding, and a switched mode psu down the right of the PCB.

The documentation

The manuals for the Laser 200 are no worse than those for many other computers, and are better than some. There's the usual smattering of spelling mistakes, most of which are harmless, and the level is pitched at the rank beginner. Unfortunately the manual has equated simplicity with brevity in many places, and a beginner may need rather more explanation of some aspects. The sample programs are all short and quite basic.

I particularly liked page 21, which had a drawing of the screen with SYNTAX ERROR displayed on it, and beneath it the explanation, "This means SYNTAX ERROR..."

BENCHMARK	TIME
BMI	1.7
BM2	7.0
BM3	17.0
BM4	17.4
BM5	19.3
BM6	31.6
BM7	48.8
BM8	72.5
Average	26.9

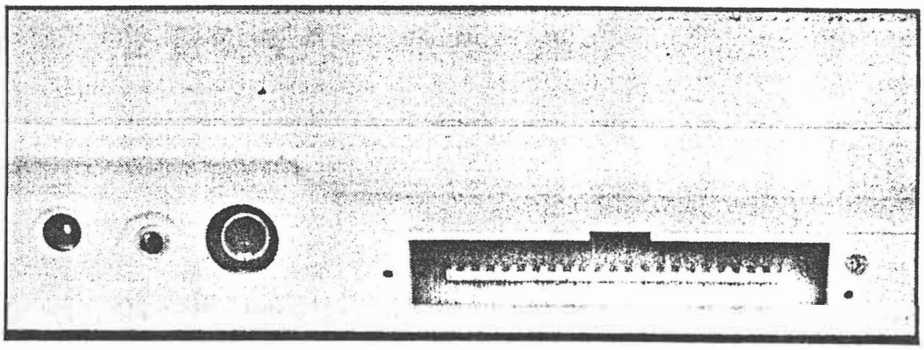
The Benchmark test results.

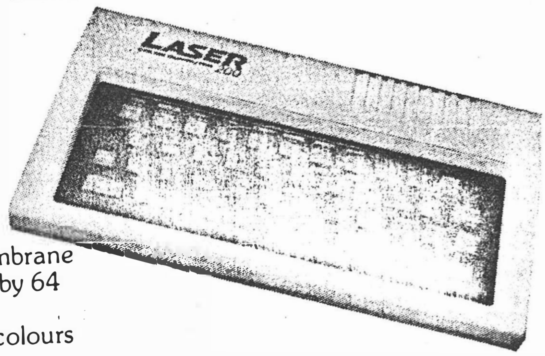
Conclusions

Throughout this review I have made comparisons between the Laser 200 and the Spectrum, which one tends to do instinctively given their similar appearance. In reality this is probably an unfair comparison, because although the Laser costs only £70, the basic computer has only 4K of memory and the price of a 16K RAM pack takes the price up to that of a 16K Spectrum, which offers much better graphics and more facilities for expansion now that the Microdrives and networking are available. (On the other hand, the 64K expansion takes the price to that of a 48K Spectrum). Perhaps a fairer comparison for the basic machine would be one made with the ZX81, another computer intended as a low-cost entry into computing but with an inferior keyboard and no sound and colour.

Unfortunately Sir Clive, with his usual consummate timing in these matters, has just dropped his price to £45 for a ZX81, 16K RAM pack and software cassette, forcing people to decide whether it is worth paying the extra £25 for sound, colour and a quarter of the memory: not to mention the vast amount of software available for the two Sinclair machines which widens the gap even further. It seems that the Laser 200 has fallen between several stools, and it may remain there unless the distributors can stimulate the interest of the commercial software houses.

Below: The back of the Laser 200 showing the memory expansion bus.





FACTSHEET

CPU	Z80A
ROM	16K
RAM	4K (expandable to 16 or 64K)
Language	Microsoft BASIC
Keyboard	45-key multifunction, moving rubber membrane
Display	Text mode: 16 lines of 32 characters, 32 by 64 pixel graphics in eight colours plus black. High-res mode: 64 by 128 pixels in four colours (choice of two sets), no text. TV or monitor output
Cassette	600 baud
I/O	Centronics printer interface, lightpen, joysticks, bus for memory expansion
Sound	Single channel, 31 notes, 9 durations
Costs	Laser 200 £69.95 16K RAMpack £29.95 64K RAMpack £59.95 Printer interface £19.95 Joysticks £19.95 per pair Lightpen £19.95

Supplier Computers For All,
Southfields Industrial Park,
30 Hornsby Square,
Laindon,
Essex
Telephone 0268 418414

Buying your first computer

Many thousands of people have bought computers for the home and perhaps many more would if they knew what they were all about. If you think you'd like a personal computer but are unsure which way to go, read on.

by PETER VERNON

Buying a computer is not a decision to be taken lightly. At the very least a small computer will set you back about $200 and that's without any of the essential accessories, programs and books that go with the hobby. A personal computer can quickly become an open invitation to spend money. How can you be sure this money is well spent?

The first step is *not* to rush out to a dealer or computer store. Instead, do some work with pencil and paper to define your own requirements. What do you want a computer to do? The expected applications of a computer define what is required by way of memory size, screen display format, keyboard and accessories.

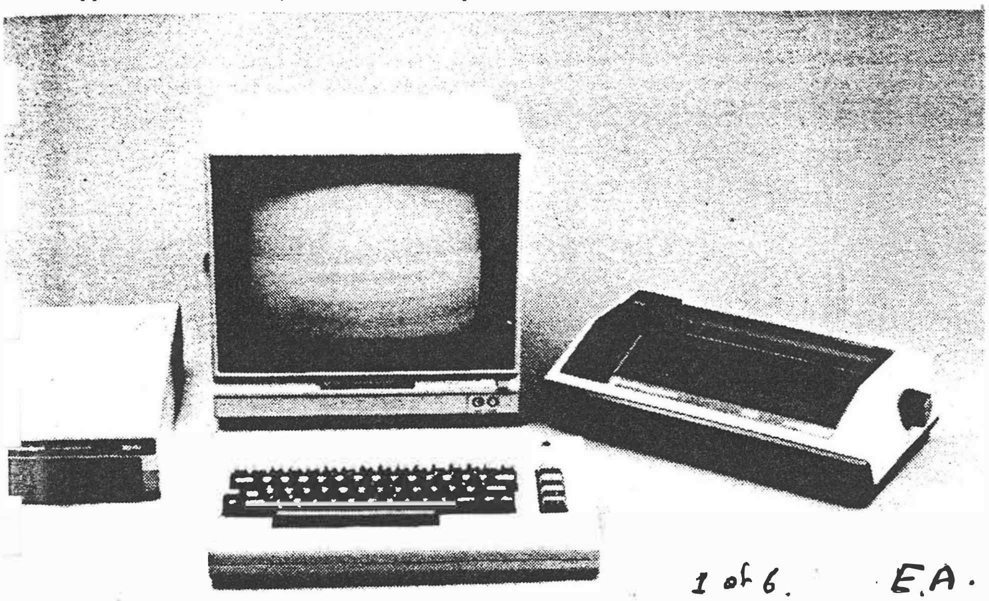
Personal computer applications

By far the most popular use of a computer in the home is to play games, with educational applications a close second. Word processing, home management, control of household appliances and communications are other uses. With the increasing number of computer clubs, owning a computer can also be an introduction to a large circle of friendly, like-minded people.

There are some things, however, which buying a computer will not do. It will not make you into an expert programmer — no more than buying a piano will make you a musician. Long hours of learning and practice are required to master anything but the simplest programming.

Nor will owning a personal computer guarantee you a job, although it can help. Some people have graduated from a computer club to working with a computer manufacturer or distributor of computers or software, but the competition is fierce and opportunities are scarce. Familiarity with computers can have indirect benefits at work, or may impress a prospective employer!

The Commodore 64 is one of the largest-selling computers in the home market. Features include 16 colour graphics, sound effects and plug-in software cartridges for games and home management applications. While it is a system with a lot of potential considerable programming effort is required to bring out the best in the machine because of the limited Basic language supplied. See EA June, 1983 for an in-depth review.



What to look for

Having decided what a computer can and cannot do for you the next step is to start looking around. Read reviews, advertisments and the brochures produced by manufacturers and retailers. Talk with other computer users but remember that they are the last place to go for unbiased advice. To most users, the best microcomputer is the one they own!

If your primary interest is games, look at the capabilities for colour graphics, sound effects, joystick facilities and available software. Although games programs are sold for computers which do not have a colour display capability, they are intended as adjuncts to other applications. Colour adds immeasurably to the impact of computer games, so if this will be the main use of your computer there is little point in going for a monochrome display.

The features required for educational uses of a computer are similar to those required for a games machine. The best educational software uses colour graphics, sound effects and interaction to maintain the student's interest. Old style "drill and practice" programs are rarely worth buying.

For word processing and information management, colour is not an essential requirement. The ability to display both upper and lower case letters and a reasonably sized text display are much more important.

The VZ-200 from Dick Smith Electronics is one of the lowest-cost systems on the market. It offers low resolution graphics in eight colours and limited sound effects but is a good starter system at around $200. The Tandy MC-10 is comparable. The July, 1983 issue of EA contains a review of the VZ-200.

The standard screen format for business word processing is 25 lines of 80 characters each. Home TV sets, even those converted for direct video entry, just do not have the bandwidth required for a legible display of this line length. It is for this reason that most low cost computers display 32 or 40 characters per line. Longer lines exclude the use of colour.

If colour is not required, a monochrome monitor or converted black and white television receiver can be used. The legibility of the display will depend on the bandwidth of the monitor. Television receivers converted for direct video entry can be used to display 64 character lines, but for a crisp 80-column display a higher priced video monitor will be required.

A video monitor is just one of the "hidden costs" of a personal computer. Prices range from around $200 to over $700 for a 34cm (diagonal) colour unit, although the experienced electronics enthusiast can save some of the cost by converting a surplus television set, as described in the August, 1983 issue of EA. As well as allowing full time use of a computer, a video monitor generally provides a sharper picture and is less prone to interference than a television set driven by an RF modulator.

Note however that some computers limit the choice of methods. Some, like the VZ-200 and Commodore 64, provide both modulated RF video and direct video output, while others such as the TRS-80 Color Computer provide only a modulated RF output, and cannot be used with a monitor unless the case is opened (voiding the warranty) and additional connections made. Others, such as the VIC-20 and TI-99/4A require an RF modulator in the form of an external box, usually supplied with the computer.

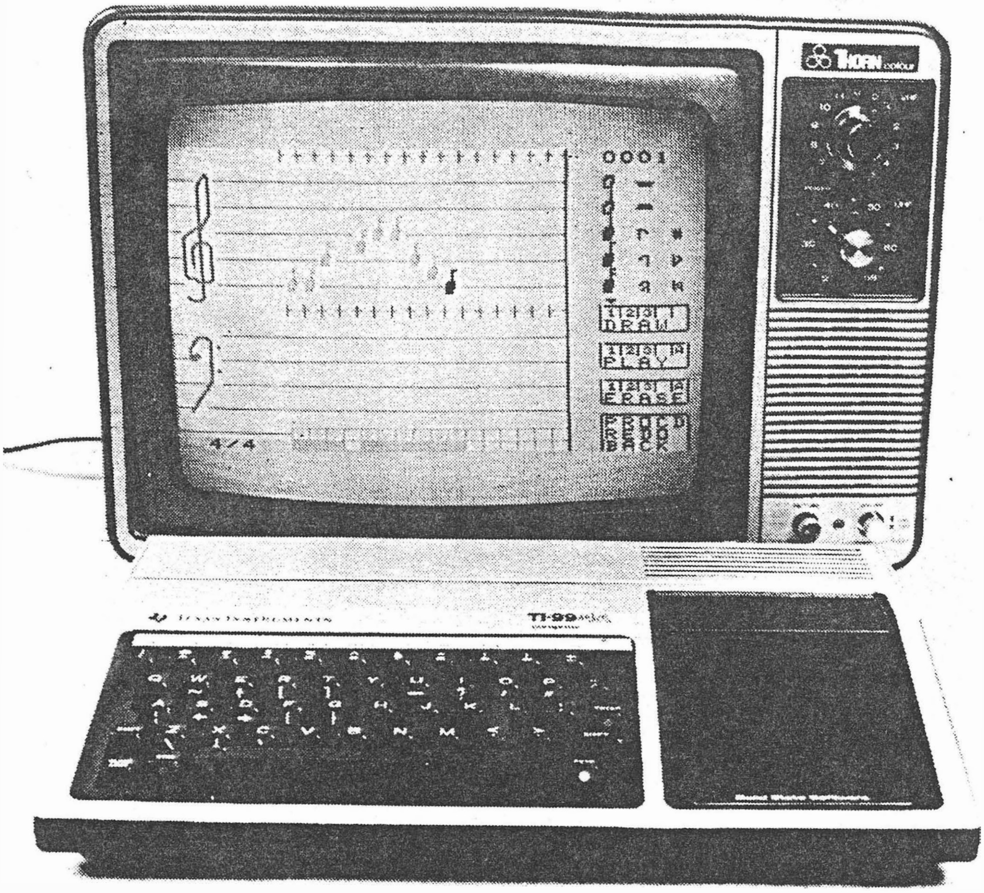
Naturally, a printer is also required for word processing applications. We won't go into the relative merits of dot matrix, thermal and daisywheel printers here, other than to point out that a printer can cost much more than the computer itself. Unless word processing is going to be you main application it is not necessary to purchase a printer immediately. Wait until the need becomes evident.

More important, however, is a place to connect the printer. A surprisingly large number of personal computers are not equipped with either parallel or serial ports but require separate "printer cards" and communications interfaces as an extra cost option. Other computers can

Tandy's TRS-80 Color Computer is available with one of two versions of Basic. Extended Color Basic is require to make use of the computer's sound and high resolution graphics capabilities. Other features include plug-in cartridge software and a range of disk operating systems.

2 of 6. EA Jun 84. p13.

Buying your first computer



Discontinued last year by TI because of marketing problems, the Texas Instruments TI-99/4A is currently available at bargain prices. Features include excellent colour graphics (including 32 sprites) and sound effects, and a very good version of the Logo language. Some software in plug-in cartridges is still obtainable at dealers and the computer is supported by a very active users group with branches around Australia. This system was reviewed in the December, 1982 issue of EA.

be used only with a printer from the same manufacturer because they use a non-standard interface. Since these printers are usually more expensive than standard types, another extra cost is involved.

If your main use for a computer is to learn about hardware and design techniques, access to the circuitry of the computer is important. Adding your own devices to a computer is one of the best ways to develop an understanding of the principles of computer engineering. If this is impossible, either because the system is not readily expandable or there is no information available on the expansion facilities, the computer is not suitable for the electronics hobbyist.

Documentation is important here, however. The mere presence of an expansion port or cartridge connector is not enough unless there is sufficient information available to allow the use of the facilities. At the very least, a description of the pin-outs of the connector and the allocation of memory is required.

Graphics — what's available

Graphics capability depends on two factors; the number of different colours which can be displayed on the screen and the resolution of the display. Resolution is usually expressed as the number of dots or "pixels" which can be displayed across the screen by the number which can be displayed vertically. (Pixel stands for picture element.) The more dots horizontally and vertically, the smaller the size of each dot and the greater the detail which can be displayed.

For systems costing less than $500, 16 colours and a resolution of 256 (horizontal) by 192 (vertical) are reasonable. Like most aspects of personal computers however, graphics capabilities can be expressed in many ways, some of them ambiguous. A reference to 16 colours, for example, always includes black and white as colours. There are also trade-offs between colour and resolution. Some computers restrict the use of colour in high resolution displays because of memory or processing limitations.

Computers such as the VZ-200, TRS-80 Color Computer and the MC-10 provide low resolution "chunky graphics" and boast eight colours. In actuality, the low resolution graphics mode only allows four colours to be displayed simultaneously, selected from one of two sets. Since the background of

The CAT computer from Dick Smith Electronics is one of the newest on the market. For $699 it offers limited Apple II compatibility and enhanced graphics and sound, while a $99 "soft emulator" is available to allow the CAT with a disk drive to run the majority of Apple II software. A detailed review of this system appeared in EA in May, 1984.

The Atari XL computer system forms a compatible range from the 600 to the 800 and up. The family is known for extensive colour graphics, ease of use and the availability of a wide range of software and peripheral equipment. The 600XL shown here comes with 16K of RAM, and is expandable to 64K.

the display must be one of these colours, actually only three colours are available from graphics displays.

The TRS-80 Color Computer has a higher resolution (256 × 192) graphics mode, but only two colours are available in this mode. Of the popular personal computers only the Atari, VIC 20, Commodore 64 and Texas Instruments TI-99/4A allow 16 or more colours with relatively high resolution.

Another factor contributing to ease of programming for games is the availability of "sprites"; blocks of graphics which can be defined and moved independently of the remainder of the display. Because sprites ease the task of creating animated displays they can allow "arcade quality" video games programs to be written, even in a slow language such as Basic. Used with assembly language, they allow effects which frequently surpass dedicated video games machines.

Music and sound effects

Sound effects add considerably to the impact of computer games, quite apart from the opportunities provided for learning music theory. Computer circuits for producing sound can be divided into two types — so-called "single bit" sound and those that use a separate sound generator chip.

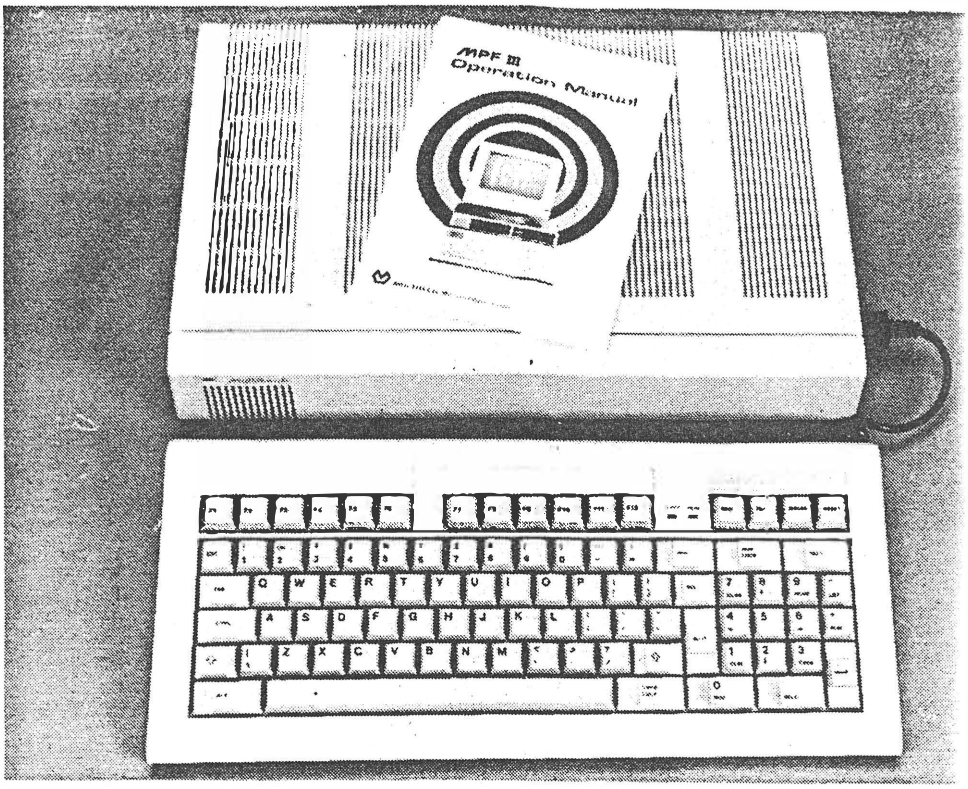
Single bit sound, as the name implies, uses one line of an output port to drive a transistor amplifier and speaker. Some systems use more than one line, however, driving and rudimentary digital to analog converter. The significant point is that the frequency and duration of the sound is controlled by the microprocessor, so all other operations come to a stand-still while sound is produced. Simultaneous sound and movement, for instance, can only be programmed with difficulty.

Computers using dedicated sound generator chips, such as the Commodore 64's "Sound Interface Device" (SID), provide a wider range of sounds, including white noise, and produce sounds simultaneously with video displays and other processing. Often the volume of the sound can also be controlled by software, unlike the single bit approach.

The other factor to be considered is the means of sound output. Methods range from incorporating an internal speaker (as in the Apple II and lookalikes) to modulating the sound onto the RF video carrier (as with the Tandy Color Computer and Commodore machines). When computers which use this method are connected to a video monitor the sound is lost unless the monitor includes a speaker and provision is made for a separate audio connection.

Few direct entry video monitors include an audio input (one exception is the Dick Smith monitor, actually a converted portable television set). For

Buying your first computer



The MPF-III provides Apple II compatibility in a compact, low profile design with detachable keyboard. Features include an 80-column display, 64K RAM, printer and cassette ports and an Apple compatible hardware expansion slot. As yet however, no colour graphics are available. The February 1984 issue of Electronics Australia has a review of the system.

this reason, computers such as the VIC 20 and Commodore 64 have a separate audio output which can be connected to an external amplifier.

Keyboards

The type of keyboard available on a personal computer also affects its usefulness for a variety of roles. Generally keyboards are of three types; flat plastic membrane switches, such as those of the Sinclair ZX81, rubber or plastic buttons (so called "chiclet" style, because of the resemblance to pellets of bubblegum), and full-stroke "typewriter" keyboards.

Flat plastic membrane keyboards are difficult to use for long periods becaus of the lack of tactile feedback. One user described the sensation as "like typing on a block of wood". In an attempt to overcome this, most such systems provide an audible "beep" to indicate that a keystroke has been registered.

Half-way between flat keyboards and full typewriter style are "pushbutton" keyboards, as used by the TRS-80 Color Computer and the IBM PCjr. This type of keyboard is easier to use than the flat type and is more suitable for applications around the home.

Apart from the style of a keyboard there are very few guidelines which can be laid down. Separate numeric keypads, while convenient on office computers used for large scale data processing, are of little use on a personal computer. Far better is a cluster of cursor control keys and special function keys which can be re-defined by the user.

As long as a keyboard is comfortable there is very little to choose between alternative offerings. Any keyboard used for more than a month tends to become a natural arrangement, and one quickly becomes familiar with various quirks and foibles.

Software

The availability and method of loading software is one of the most important aspects of a computer to be used in the home. By far the best method is the solid-state ROM cartridge, which avoids the problems and delays caused by loading a program from disk or cassette.

Most of the popular low cost computers for home use are designed to accept program cartridges, but cartridges

The Australian-made MicroBee computer has attracted a lot of attention from home and educational users. This is one of the few low cost machines to offer text displays of more than 40 characters per line (almost essential for word processing) and is supplied with a range of software. The photograph shows the start-up menu of WordBee, the MicroBee's built-in word processor. The MicroBee IC model was reviewed in EA in November 1983.

5 of 6.

Buying your first computer

intended for one type of machine are not transferable to another. The range of programs available in this form may also be limited, so it is best to assess the variety and cost of program cartridges available for a particular computer before committing yourself to a purchase.

Other programs may be distributed on disk or cassette, and in any case you'll need some form of "mass storage" to permanently retain copies of your own programs and data. The lowest cost method is to use a standard cassette recorder. Disk drives are faster, but more expensive, and are better left until you have some experience with the computer and come to feel the need for faster response time and greater storage capacity.

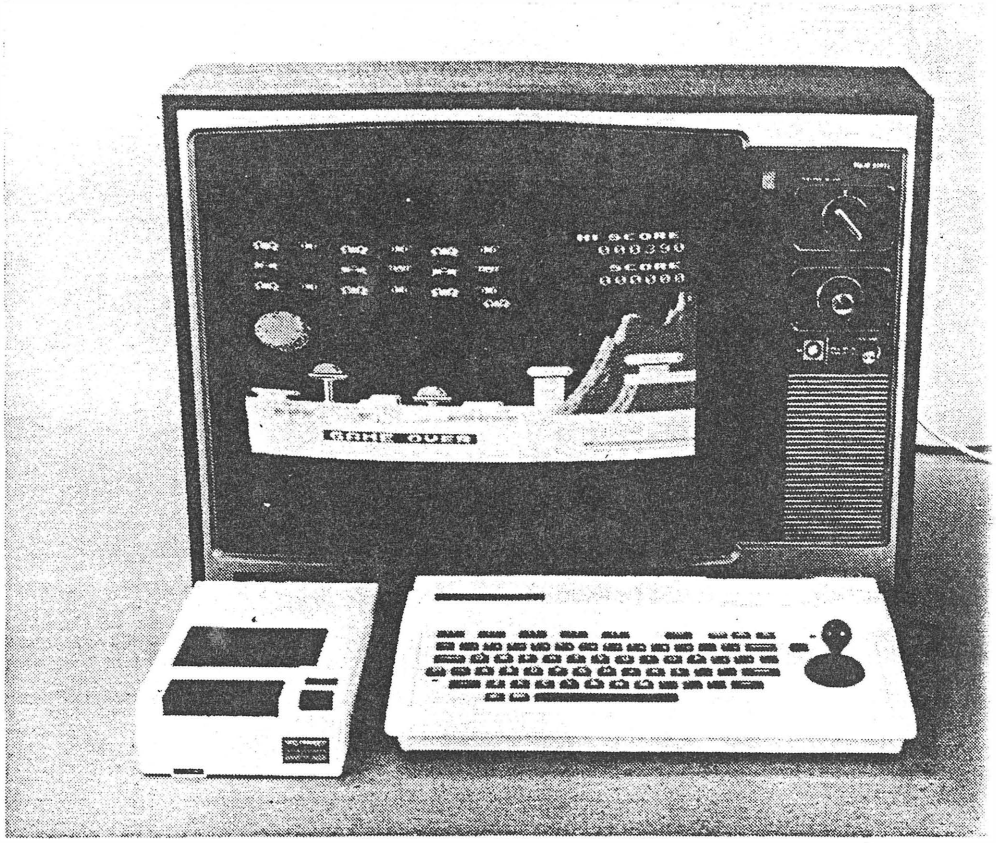
Be aware that some computers cannot use an ordinary cassette recorder. They require a specialised device made by the computer manufacturer and often selling for twice the price of a budget portable cassette player. The Commodore VIC 20 and 64 and the Spectravideo machines follow this practice. The VZ-200 also requires a comparatively expensive cassette player for reliable performance.

Memory size and the choice of a processor

Surprisingly, the size of a computer's programmable memory is not as important as it first appears. Most personal computer systems are provided with enough memory for typical applications, while those using software ROM cartridges can run programs without reducing the size of RAM.

Because of the low cost of dynamic RAM chips, memory sizes of 16K are most common, with expansion in increments of 16K. One "K" is 1024 bytes or characters, but because Basic programs are usually stored in a compressed form, more space is available than would first appear. Machine language programs, of course, are even more compact.

16K of memory is adequate for programming and educational applications. Word processing may require more, as a single typed page may contain around 2400 characters, limiting in-memory storage to around seven pages of text in 16K.



The Spectravideo SV-318 offers colour graphics and sound effects and a built-in joystick for $399. The graphics mode features 32 "sprites" or patterns which can be defined and moved around the screen independently of the remainder of the display, easing the task of writing fast-moving games and other display programs. The larger SV-328 does not have the built-in joystick but offers a larger, full-stroke keyboard. See EA for February 1984 for a review of the SV-318.

As important as the absolute size of a computer's memory is the use made of it. All computer operating systems require some RAM for storage of temporary variables and for display memory. What matters is "usable memory", which can be quite different from the total memory advertised. The Commodore 64, for example, is commonly advertised as a 64K system, but in fact only around 31K is usable from Basic.

The important distinction to be aware of is the difference between RAM and ROM. RAM, or Random Access Memory, holds the user's programs and data. ROM holds the computer's operating system and (usually) a Basic interpreter.

Read Only Memory size can vary between two models of the same computer. Many machines, such as the TRS-80 Color Computer or the TI-99/4A, offer two versions of Basic, one standard and the other an extra-cost "Extended" version which is required to make effective use of the computer's graphics and sound capabilities. You should be aware of which version you are getting for your money, as the most advanced facilities are usually only available with Extended Basic.

So far no mention has been made of the varieties of microprocessor chips which form the basis of all personal computers. There is a good reason for this — if a computer has the capabilities that you want, it doesn't matter which microprocessor it uses. Debates on the merits of the 6502 or Z80, or 8-bit versus 16-bit processors are irrelevant to the actual applications of a computer in the home. If a machine does the job that you want it to do, what more can you ask?

In conclusion

A computer console, cassette recorder and a television set are enough to get you started in personal computing. It is only a start, though not the end of the road (or the expense). If you intend to keep up the hobby, look for an expandable system which is well supported by software supplies and publishers.

Consider joining a computer user group, possibly even before you purchase your own computer. As a source of advice and assistance, for a subscription of around $20 per year, such groups are worth their weight in microchips!

In the end however, the decision on what sort of computer to buy is your own. The more time you put into defining your own requirements and applications, the easier the final choice will be.

6 of 6.

An important role for

"If you're planning to invest in a computer, buy a real one, not a toy!" That's the kind of advice you're likely to get from a computer buff — but it may be a rather one-eyed opinion. The fact is that some of those "toys" can provide the means and the incentive for beginners in all age groups to learn the elements of computing in a pleasant and not-too-expensive way. It's worth thinking about.

FORUM

Conducted by Neville Williams

In the wake of that rather positive assertion, I should perhaps qualify my earlier remarks in "Forum" for November '83, under the heading: "Do computers really have a place in the home?" While it contained a passing reference to the tuitional value of a domestic computer, in the main, the article tended to question the relevance in the average home of a complete system: computer, monitor, printer, disk store and so on.

The message that came through was one of caution: think carefully before you talk yourself into spending a couple of thousand dollars: it could turn out to be a very poor investment, if you have no real use for it.

This time around, we are talking about a purely tuitional role for a small computer within the family unit and an outlay of between $100 and $200 — a tiny fraction of the earlier figure.

Much the same qualification would apply to Peter Vernon's article in the June '84 issue: "Buying your first computer". He talks at length about basic computers, monitors, printers, memory stores, software, etc — all of it directed at would-be computer buffs who intend, ultimately, to acquire a complete system costing thousands of dollars.

I repeat: that is not what I have in mind in this article.

What follows was prompted in part by a letter to hand from a reader in Willunga, South Australia. He says:

Dear Sir,

For some years I have been considering buying a personal computer, mainly so that my children can acquire some familiarity with this burgeoning discipline.

My worst suspicions were confirmed in your review of Ian Reinecke's book "Microcomputers" (EA April '84, page 110) in which I read ... "a vast difference between low cost machines ... compared with a machine costing several thousand dollars, which is needed if any real use is to be obtained". And later ... "be better off (purchasing) a set of encyclopedias".

Contrast this with Dick Smith's latest catalog where the Editor of Personal Computer magazine is quoted as saying of Dick Smith's $169 VZ200 ... "I'm certainly going to buy one".

Where then is the truth?

Are the cheap personal computers with, say, a 16K or 32K memory of any real use? How much use? Are people buying them only to play Pacman? Or are they a real instructional tool?

Your response will guide my buying decision.

A.T. (Willunga, SA).

Understandably, correspondent A.R. is worried by the apparently opposite opinions expressed by author Ian Reinecke and the editor of "Personal Computer" magazine. One talks about buying a computer which the other would apparently consider to be of no real use (hence the heading to this article).

Reportedly, Ian Reinecke makes two particular points:

- For most serious applications, forget about low-cost "machines", intended primarily for playing electronic games. To be of any real use, the equipment would be quite costly, eg "several thousand dollars".
- Appropriate educational software is very limited: "the whole subject is really a joke".

I am not in a position to debate his opinion of available software but his observation about equipment is not at variance with what was said in "Forum" or in Peter Vernon's article, mentioned earlier.

If parents want to set up a computerised educational system in the home, it will need to approximate the system which students encounter at school/college; that means at least MicroBee or Apple or other such equipment, costing two or three thousand dollars all up. It would have little in common with "low-cost machines ... mainly intended for playing games".

If we thus appear to support Ian Reinecke's ideas about equipment, where does the humble VZ200 fit in? Is it indeed a toy; of little real use?

In reality, the DSE VZ200 may not have been considered when Ian Reinecke's book was written and it may be in a class somewhat above his despised games-type "machines". Even so, it may still not have earned his approval, being considered too far down-market to form the heart of a serious system.

Many would share that view.

In fact, the VZ200 is not primarily a games machine. For sure, one can set up and play games on it, as with most other micros, but beyond that it offers, in terms of our own review in the July '83 issue: "colour, a reasonable amount of memory and a powerful Basic interpreter".

We concluded our review in the following terms:

"If you want a computer to look after your share holdings, or for word processing, look elsewhere. If, on the other hand, you want a computer for playing games, for self-education, for learning about Basic and perhaps for writing your own programs, the VZ200 has one overwhelming advantage — the number of features for the price."

At the time, a practical computer for under $200 was a real price breakthrough comprising, as it did, the basic unit, power supply, cables and a comprehensive manual. Having in mind our own reaction, it is not really surprising that the Editor of "Personal Computer" should have decided that he had good use for just such an item — for the kind of secondary reason which we ourselves suggested.

If the reasons were valid at $199, they would be attractive at the subsequent price of $169 and positively compelling at the latest figure of $99.

As a matter of interest, I questioned

 1 of 4

'useless' small computers!

From the Commonwealth Employment Service:

Dear Mr Simpson,

I was impressed by Neville Williams' "Forum" column in the April '84 issue of Electronics Australia: "What do you do when you can't find a job?" I would like to include the article in information available to clients of the Career Reference Centre.

The Centre, which is operated by the Commonwealth Department of Employment and Industrial Relations, provides a free occupational information and vocational training information service.

My intention is to include clearly sourced photocopies of the article in our job information folders on occupations related to electronics and job seeking skills.

I request your permission to photocopy the article for the purposes described.

Paul Mitchell,
Manager, Sydney
Career Reference Centre.

Ike Bain, Managing Director of Dick Smith Electronics, as to the reason for such a dramatic price reduction. He nominated two factors: economy of scale in manufacture and fierce worldwide competition between computer makers.

Hopefully, A.R. of Willunga should by now have glimpsed a glimmer of light at the end of the tunnel.

If he has in mind a complete computer educational system, comparable with those in schools and colleges, then it is going to cost him "X" thousand dollars, as per "Forum", Peter Vernon and Ian Reinecke.

But I don't really read that requirement into his words: *"so that my children can acquire some familiarity with this burgeoning discipline"*.

If his prime objective is to create a familiarity with computers at a family level, and to dispel the mystique which faces the uninitiated, young and old, then he can accomplish that and move on to a working knowledge of programming for a much more modest figure; like $99 for example!

In fact, that's exactly what I want to talk about from here on.

On two separate occasions, recently, I have been the involuntary witness to a family argument — sorry, discussion — during which the children were trying to convice their father that he should buy a computer for them to use at home:

"But, Dad, you can get one for less than $200 ... go on Dad!" (This was before the most recent price reductions.)

In both cases, the father insisted that there was more to it than that. You couldn't do much with just a keyboard and, by the time they had bought all the stuff to go with it, he'd be up for nearer $2000! Right now, he didn't have that sort of money to spare!

To see kids of high school age arguing for a computer was no surprise, because computers are now a part of the high school scene, but the 7/8-year-olds were joining in with hardly less conviction. Nor was there any special mention of electronic games. It was simply: "buy a computer, Dad!"

Watching the performance, I couldn't escape the impression that the kids were really asking for a contemporary learning tool, much as in other days, when we wanted our own slate and slate pencil (!), our own box of water colours, our own drawing instruments, our own slide rule, our own calculator. Now they want access to their own computer and the opportunity to gain an easy familiarity with the machine that, more than anything else, typifies their kind of world.

Perhaps they don't need to store or print out, to process words or to keep accounts; that can come later. Maybe their first and urgent requirement is to come to terms with the keyboard, with computer language and procedures; to do a few exercises, work out a few problems, observe some basic graphics and play a few games routines, all as part of the learning process.

Nor is the need to learn unique to children. Adults also must adapt to the world of keyboards which has been created by their own generation. Here I could quote Professor Brian Garner, head of computing at Deakin University in Geelong (Vic) and Chairman of the recent Computer Data 84 Conference in Sydney:

"Parents will have to learn about new technology and how to use it or they will be left behind by their children.

"Parents should spend more time with children and share their involvement with computers."

If they fail to do so, Professor Garner warned, stress will tend to develop between computer-literate children and parents who have no understanding of the new technology.

Seeking to probe the computer-awareness of present-day high school children, I have been asking a few questions on the subject lately, whenever the opportunity presented itself.

An English subject mistress professed to know little about computers but had her own reason to be impressed: students who had access to home computers, she said, and especially to word processing facilities, had re-developed the long-lost art of checking their work before handing it in!

"It has changed their attitude to detail. They hand in better work and are rewarded by higher marks."

A maths master from another high school said that students generally were aware of computers but actual knowledge of them ranged all the way from minimal to those who had earned the right of access to school computers without teacher supervision.

"Some of these kids are really good."

Could he see a role for a simple computer in the home, purely to allow children and parents alike to learn the rudiments of the subject?

"Most decidedly!"

Another high school maths teacher obviously shared these opinions but added that he did not much mind if students spent some of their free time setting up their own games routines. Games or no, they were still learning how to program, and doing so with added incentive and concentration.

The manager of an electronics store confirmed my teach-yourself ideas in a moment of personal frankness:

"When I accepted this job, I was literally scared of computers. But I took a small one home and spent a couple of weeks working through the manual. I'm still a beginner compared with some of the kids that come in here after school but, at least, I now understand what they're on about!"

As a matter of further interest, I posed the question to an executive of Dick Smith Electronics:

"Why do people buy your VZ200?"

"For all sorts of reasons", was the reply "but we tend to emphasise its value as a means of self tuition. Look at our catalog:

"Getting left behind in the computer race? Here's the solution ...

2 of 4.

"Bring your kids into today's technology . . .
"Easy to read manuals . . .
"Learn fast . . . and so on".

Never a company to miss a trick, DSE responded further to my question with the invitation to try it for myself, and with a carton containing a VZ200 on loan, along with extra memory module, cassette recorder/player, printer, interface, typical software tapes and assorted manuals.

I was happy to take up the invitation but I left the peripherals in the box, primarily because I wanted to sample a tuitional exercise involving just the basic $99 computer and, at most, one or two of the supplementary manuals. Such an exercise would not be entirely fictional, because I had never before handled the VZ200 and literature and, unlike Peter Vernon and Co, I tend to get rather rusty between spaced-out exposures to computer whatnots.

What were my reactions?

While the VZ200 has a keyboard conforming nominally to QWERTY (typewriter) layout, it uses "rubber" pads rather than full-travel keys and provides for upper-case (capital) letters only. There is no space bar, as such, the function being handled by a space key at the lower right-hand corner.

> ***Teach yourself to drive the VZ200 and you'll have little difficulty in adapting to other Basic-language micros***

The pads present no great problem but one has to overcome the tendency to type as if normal lower and upper case letters were available — and in the process, tapping the lower lip of the case instead of the non-existent space bar!

While these very characteristics limit the potential use of the VZ200 with a full-scale system, they are of little consequence at a tuitional level. More importantly, the keys give user access to a powerful — and normal — programming facility in computer Basic language, plus colour graphics, and more, if advantage is taken of it. Teach yourself to drive the VZ200 and you'll have little difficulty in adapting to other Basic-language micros.

Packaged with the VZ200 is a small user manual, a booklet containing 20 programs, a demonstration cassette, and a 166-page instructional manual produced by the manufacturers in collaboration with Jamieson Rowe, the former editor of this magazine. The manual begins with the question "What is a computer?" and proceeds on a step-by-step learn-while-you-do-it basis to introduce simple calculator functions, a wide range of computer routines, colour graphics and "music", with appropriate references to the possible use of an ancillary cassette deck and printer.

Other instructional manuals available for the VZ200 include "Introduction to Computing" by Toni Louise Henson and "Getting Started" by Tim Hartnell and Neville Predebon. Both are written in friendly, casual style which helps turn the learning experience into relaxation rather than a chore. Either or both can be used in conjunction with the manufacturer's manual to pick one's way through the various keyboard routines.

If you ultimately decide to spend $99 and to repeat the exercise, your memory may or may not cooperate as you are introduced progressively to the special significance of certain punctuation marks, instructions like BREAK, RETURN, GOTO, GOSUB, etc, and to statements like IF-THEN, FOR-TO-NEXT and so on.

If you can remember them, fine! But don't get discouraged if you seem to keep on forgetting them; having to rely on the manuals or your own scribbled notes. It's not supposed to be a test of memory but an exercise in reading and doing — and seeing it happen for you, in your own home, on your own computer.

More importantly, as it does so, the "faze" and the mystique will begin to drain away and interest will quicken. You may even feel somewhat miffed when the family wants their TV set back to watch the news or "Country Practice". Maybe you will have just accomplished your first bit of solo programming by turning Toni Henson's "What Number" exercise into a genuine random number repetitive game!

If you want to pursue the exercises to a genuine facility at the keyboard, two complete books of programs are available for the VZ200, before venturing further afield. But, by this time, you may have developed into a computer nut, anyway!

You may never reach that stage but that's really of secondary importance in the present context. What matters is that, somewhere along the line, you will have ceased to be afraid of keyboards and computers. You will have had the experience of driving one and come to realise that the essential difference between fear and facility is time and practice.

For you, and possibly for other members of your family, the exercise will have been justified.

At least, that's the way I saw things, following my own simulated exercise.

Is the DSE VZ200 the only option by way of an inexpensive tuitional computer?

No it isn't.

While preparing this article, I paid a visit to the local Tandy store and posed the question:

"What's your answer to the Dick Smith VZ200 as a stand-alone tuitional computer?"

> ***What matters is that, somewhere along the line, you will have ceased to be afraid of keyboards and computers***

The attendant's reponse was to direct my attention to something I had already noticed on entering: a display featuring the Tandy TRS-80 MC-10 personal computer, marked down from its original price of $179.95 to $99.95.

Why the huge reduction? Is it being discontinued? A clearance sale?

No, I was told, that would be the continuing price, thanks to worldwide competition in the computer industry.

The Tandy MC-10 is physically smaller than the VZ200, with less memory (4K) and probably somewhat less versatile programming. But it does have a space bar and keypads with agreeable tactile response, plus output ports for tape deck and printer. It comes complete with mains power supply, cables and instruction manual and, while there is less other off-the-shelf literature, Tandy told us that is is supported by an independent users club.

From what we could judge by looking at the package in the store, it too would offer a useful tuitional facility for under $100.

In the same week that we visited the store, Tandy were offering $100 off the price of their standard keyboard models, bringing the price of their base model to a temporary $249. That would probably represent a greater outlay than many would be prepared to write off as a tuitional exercise but it does indicate the way that computer prices have fallen during the last 12 months.

Who knows what readers may be able to pick up by way of a tuitionalcomputer, over and above the VZ200 and the MC-10? Just make sure, however, that it offers adequate BASIC language facilities, certainly not less than 4K of built-in memory, a mains power supply,

an RF converter to feed an Australian standard TV receiver, and a good tuitional manual appropriate for the particular model. Colour graphics and "music" are less important but, after all, they are part of the familiarisation process.

What of the peripherals you can buy to go with the VZ200 or MC-10: extra memory, B & W or colour monitor, cassette deck, printer etc? To this point, we have assumed that learners will use an available TV receiver as a monitor and, possibly, an available cassette deck, thus avoiding any extra outlay.

After a few weeks, or months, and having become familiar with the rudiments of computing, you will be in a better position to decide which way you want to go: avoid further expense, add elementary peripherals to an elementary keyboard, or plan towards a serious system for whatever purpose.

If the last named is your choice, then best you consider that your elementary lessons have come to an end. Turn back to Peter Vernon's article and start reading, thinking, acting and spending like a genuine computer buff!

Job opportunities

At this point, I would like to revert to the subject of job opportunities in the electronics industry for young people, as discussed in "Forum" for April '84. Perhaps it may not be as unrelated as it may seem, because we have just been discussing a way in which some young people may be able to add to their potential job skills.

Among the personal observations, phone calls and letters on the subject of youth unemployment, it was gratifying to receive the one in the accompanying panel, from the Sydney Career Reference Centre of the CES. It might suggest that some of the remarks in the April "Forum" were along helpful lines, criticism notwithstanding.

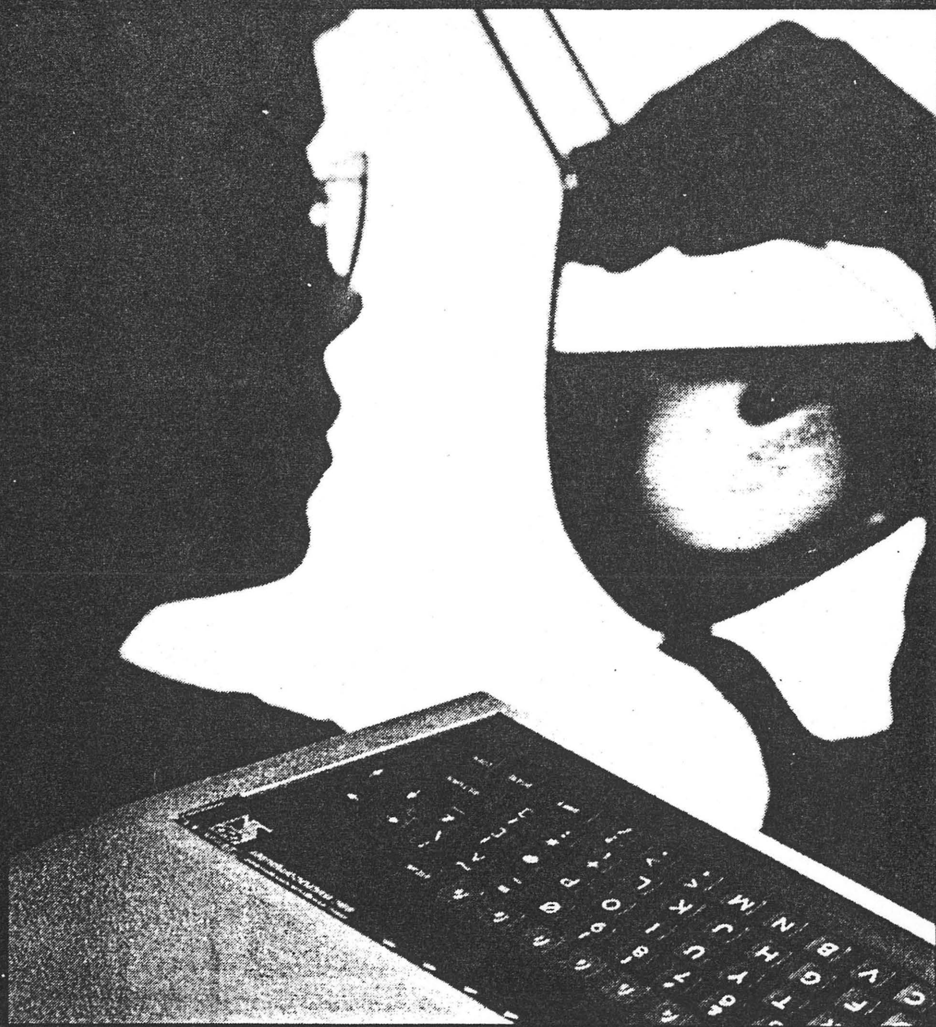
As might be imagined, Editor Leo Simpson was happy to grant permission for the article to be reprinted, with due acknowledgement to the source, and I guess that the same release of copyright would apply to other organisations or educational groups who may find the particular article helpful.

In fact, some correspondence on this subject is still outstanding but there is a limit to what can reasonably be accommodated in three pages or less, per month. Unfortunately, while the subject may become tedious, it certainly won't lose its topicality.

Even the most optimistic of politicians wouldn't try to tell us that!

Curtis Bollington looks at design and comfort

It's tough



In the Hunchback of Notre Dame's day there wasn't the choice of micros around that you have now. That's probably why he ended up hanging perilously from the gargoyles of the great Paris cathedral, wild-eyed, shabby clothes hiding his contorted frame, his mouth twisted into a set snarl and his hair matted. Whatever machine he had *definitely* didn't suit him.

You've probably come across micro users who are in more or less the same state. But, with many popular home micros on the market, you should be able to choose one which will suit your needs.

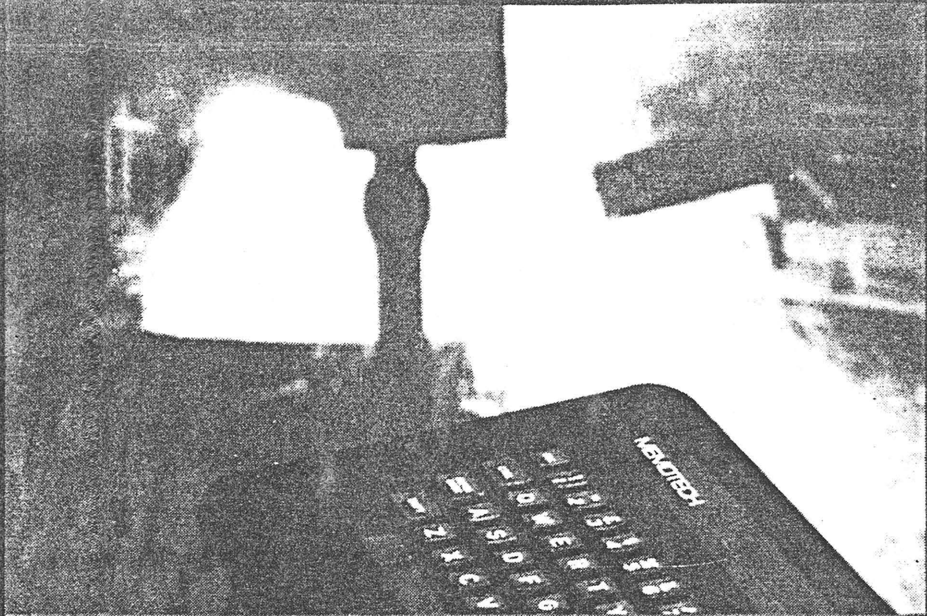
PC Games has been taking a critical look at various aspects of these popular home machines. To date we have appraised the Basic language and the sound capabilities of each micro. This month we compare their designs, documentation and ease of use.

The **Spectrum** is a computer from the Sinclair stable. It has the luxury of colour and a set of flashy rubber keys. The clever people at Sinclair must have decided that the keyboard of the ZX81 wasn't good enough to be repeated on the Spectrum. The rubber keypads are marginally better, but still nowhere near good enough for typing.

Single key entry is used with as many as five functions per key. There is a strip connector on the back of the machine for peripherals such as the printer. The TV, MIC and EAR sockets are also situated on the back of the machine.

The Spectrum measures 9 by 5.6 by 1.25 inches. The casing is substantial, being made of quality plastic. The manual supplied covers setting up the machine and the Basic tutorial. The Spectrum has forty keys. There are better quality keyboards available for the Spectrum but these will set you back around $120.00, half the cost of the 16k machines. As it stands the keyboard won't allow you to use the Spectrum for word processing.

PCG Oct 84 1(3) 82-87 1 of 5.



at the top

The VIC 20 is the cheapest micro to have a proper QWERTY keyboard. Don't worry about the term QWERTY, it simply describes the way in which a typewriter keyboard is arranged; the first six letters on the top line of characters are QWERTY.

But if you were to choose the VIC 20 to use for word processing you would be making a mistake. Although a proper keyboard is fitted there are no lower case characters — only capitals. There are 66 keys in all, including a full width space bar.

The VIC 20 is much larger than the 'pancake' design of the Sinclair machines. It stands about 2.75 inches from the surface on which it sits. This poses something of a problem if you have to use the keyboard for any length of time. There isn't enough of an area in front the the space bar for resting your wrists on while you type. Your hands have to hang over the keyboard and could become tired after a while.

There is a warning light on top of the machine to let you know whether the machine is on or off, very useful. Power is supplied from a separate transformer, a fairly hefty device which plugs via a lead into a DIN socket in the side of the VIC.

The panel which contains this socket holds a socket for the control port and the power switch. Everything is clearly marked so there's no danger of plugging things into the wrong place.

There are several ports located in the back of the machine. None of them are labelled. Referring to the manual will tell you the following. The gaping hole is for memory expansion. The DIN socket is for connecting a disk drive. The strip connector to the right of this is for the cassette and the final strip connector is a user port for a modem and other such devices.

The VIC 20 has its own cassette recorder which runs at a slightly different speed, so you cannot use any old tape recorder with it. Commodore chose this system so that you have to buy their cassette machine, which is a nuisance because it costs $49.95: a cheap machine can be used with most other home computers.

The manual supplied with the VIC 20 is designed to be easy to use. It isn't. It's confusing, messy and very off-putting.

The **Commodore 64** is the VIC 20's big brother. It looks very similar apart from the colour. (The CBM 64 is a sort of mushroom yeuk colour, the VIC is cream yeuk.)

The 64 has an extra joystick port on the side and there is a TV connector socket which allows the use of a standard cable so there's no need for an adaptor. There is also a channel selector, which is used to select which TV channel you want the computer screen displayed on.

The manual which is supplied with the

PCG. Oct 84 1(3) 82-87 2 of 5.

CBM 64 is a considerable improvement on the VIC's and it goes on to discuss Basic. This starts with 'editing' which could be confusing to an absolute beginner. It seems that if you really do want to get into Basic on the 64 you would be better buying one of the many introductory books on the market.

The **Sharp MZ-700** is another machine with a cheap look about it. There are five function keys above the main block of keys, four cursor keys to the right and the Delete and Clear keys safely out of the way in the top right-hand corner.

This machine is unusual in that it has a built-in cassette recorder and printer-plotter. The cassette recorder has a counter, which is always useful to help you find out where you are in a program and there are the usual five cassette recorder keys. To the left of this is the printer with a paper feed button, a reset light and a pen change light. The cover of this printer slides off easily to reveal the plotting mechanism: four minute ballpoint pens. The lid has a paper tear edge, which is useful because the printer only takes roll paper of a four inch width.

The back of the machine contains all the interface ports, a two pin socket for the mains lead (the transformer is built in), a reset button and a volume control. Beside the volume control are two ports which have metal plates screwed over them — one for a printer and an Input/Output bus for disk drives and other peripherals.

There are three outputs to various types of screen — TV, monitor and RGB monitor. There are two jack sockets for a cassette recorder, labelled READ and WRITE, which are the same as EAR and MIC sockets found on other micros.

The Sharp is rigidly constructed in plastic an eighth of an inch thick. It is a large machine measuring 12 by 17 by 3.5 inches.

The manual is also a large affair, and is very easy to read. Everything is explained in very simple terms. It covers what Basic is, an introduction to programming and carries on through to a fairly detailed technical manual at the back which explains all the machine's functions.

The **Spectravideo** looks superb. It has a clean cut futuristic look. It has 89 keys, including 23 on a separate calculator type keypad. The keys are proper typewriter-type keys. Unfortunately, they are stiff and spongey, and will make your fingers ache after a while.

There is a cartridge slot with a hinged door on top of the machine. It is odd that the two strip connectors on the back of the machine aren't labelled, because the joystick ports, off/on switch and power socket on the side of the machine are all labelled. Again it's a case of having to refer to the manual.

The manual is similar to the Commodore manuals — not very well laid out and confusing to wade through! You will need an alternative manual if you want to get into programming.

The ports in the back of the Spectravideo are an expansion port for peripherals, a cassette input-output port and an RF port for a lead to a TV. The RF port is a DIN socket into which is plugged a modulator.

Considering the size of the keyboard, the Spectravideo is a compact machine. The keyboard measures 14 by 4.5 inches. The entire machine measures 15.75 by 8.75 by 3.25.

The **Memotech MTX512** has a much more serious look about it than any of the other machines reviewed. It has 59 keys on the main keyboard, 12 keys on the calculator keypad and eight function keys, making a total of 79 keys in all. The keys have a firm but light feel, adding to the overall professional image.

Unlike any of the other popular home micros the Memotech has a metal case, with a black satin-like finish. All of the ports are squarely concealed in the back of the machine. There are spaces for two RS232 interfaces which aren't fitted, a monitor output, HiFi output and a DIN socket for the external power supply. A pin connector enables a parallel printer to be fitted. There is a TV socket, MIC and EAR sockets for a cassette recorder and two standard joystick ports. Concealed beneath a clip-on cover on the left hand side of the machine is a strip connector for expansion and peripherals.

The Memotech is 19 inches long by eight inches wide and a little under 2.5 inches deep. It is by far the most elegant-looking machine of the bunch.

The manual is a hefty volume which makes an attempt at being chatty but fails. It covers setting up the machine, a Basic tutorial and using a printer — but it is all difficult to wade through.

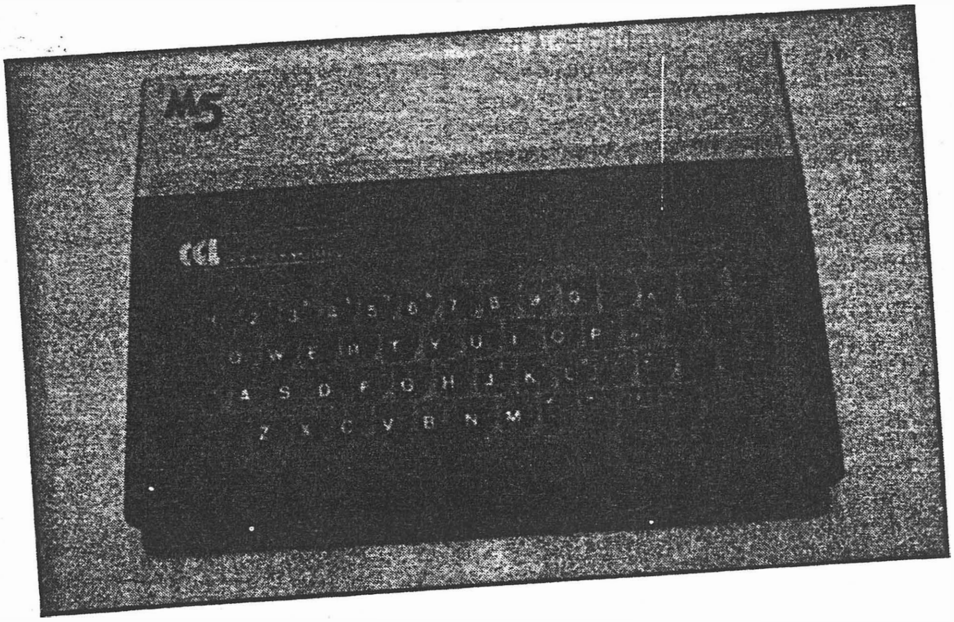
The **Tandy TRS-80** colour computer has a very practical look. There's space enough behind the keyboard on which to sit a monitor. The keyboard has a light feel, with a soft click as you press the keys down. They are set very low into the micro, which is fine except that the edge of the case hampers use of the space bar. There are 52 keys in all with no function keys or separate calculator keypad.

The Tandy has a built-in transformer and the mains lead hangs out of the back. There are two joystick DIN sockets, a serial input/output DIN socket for peripherals and a cassette DIN socket. Also on the back of the machine are a TV led socket and a reset button. The side holds a covered slot for cartridges.

Three manuals are supplied with the machine — an operation manual and two Basic manuals. One of these explains how to get started with Basic, the other is a more advanced manual for extended colour Basic. The manuals are well laid out, and, unlike those for the Memotech, are a good beginners' guide. There are indexes at the back and several program listings which illustrate features of the machine.

The **Atari XL** micros seem to be an attempt by Atari to make their micros as anonymous as possible. The old 400 and 800 definitely had more character.

The 800XL is cased in cream and brown plastic, the camouflage of suburbia, (which after all is where most micros go). There are 50 keys in all on the main keyboard which is nicely angled for ease



The Sord MS has rubber keys, similar to those of the Spectrum

 Oct 84 1(3) 82-87 3 of 5.

of use. A further five keys are disguised as a metallic strip running down the right hand side next to the keyboard.

A slot which takes the Atari cartridges is situated on top of the machine. On the back there's a TV socket and a DIN socket for a monitor. A strip connector labelled 'Parallel Bus' sits in a recess next to the peripherals socket which resembles the two joystick sockets on the right hand side of the machine in shape but which is larger.

Several manuals are supplied. 'Atari Basic' is a multi-lingual guide to Basic which is of absolutely no use to the beginner. It simply explains all the commands. The other manual is a guide to setting up the machine. Alternative manuals will definitely have to be bought.

The **Sord M5** looks horrible. The case is in yeuk-cream and dirty blue plastic, reminiscent of the interior of one of those old Cortinas.

The keys are rubber pads, easier to use than those on the Spectrum, but they flop and float around in a really insipid way.

The hinged lid above the keyboard which conceals the cartridge slot falls off easily when raised — and clips back on just as easily. When raised this lid displays instructions for operation of the keys and for loading a Basic program from a cassette tape. Given that there is a cheap stand-up lid at all, the instructions at least will be a handy help for beginners.

There are several sockets on the back of the machine — DIN sockets for the external power supply, cassette player and joysticks or games paddles and then, three small phono sockets for sound, video and a TV.

The **M5** is larger than a Spectrum, measuring 10.25 by 7.25 by 1.25 inches. The manual explains how to set up the machine and something of the functions it has, which aren't many.

The external power supply is something to behold, it measures 7.5 by 2.5 by 2.25 inches, which is a little bit over the top.

The **Electron** has 56 keys which cover letters and numbers plus a couple of extra symbols. It uses a single key entry system for Basic commands in a similar way to the Sinclair and the Sord.

The case is made of plastic with a textured finish. The Electron looks good mainly because the keyboard is the same colour as the case.

There is only one connector on the back of the Electron. This is a strip connector to which peripherals can be attached. There is no built-in power supply; the separate supply is combined with the plug. The Electron is very practically designed.

There are two manuals supplied with the Electron. One is a reference guide which covers setting up the machine, Basic and Assembler. A separate book provides an introduction to Basic which is a better guide than the other official-looking manual supplied.

The **VZ-200** is tiny. Smaller than a telephone directory (11 inches long, 6 inches from front to back, with a height of just one inch at the front of the keyboard, rising to two inches at the back), the unit is built from cream plastic. The computer is light, but does not feel excessively fragile.

The keys are rubber (much like the Spectrum keys), in light brown, with easy-to-read white legends on them. A red LED in the top right hand corner of the keyboard lets you know the machine is on (and the on/off switch is located under the 'lip' of the keyboard, down the right hand side, in a position where it would be almost impossible to turn it off accidentally).

Each key has one or two things written on it, generally a letter (the computer works all in upper case on the screen) and a symbol (such as & or *), or a graphics element.

This single element on the VZ-200 shows the influence of Sinclair. The VZ-200, however, does not demand you use the single-touch keys. If you feel happier typing out words in full (which is almost certain to be the case if you decide to move from another computer to the VZ-200), this Dick Smith machine will allow you to do so. You can even mix single-touch entered words, and spell out words, in the same program line.

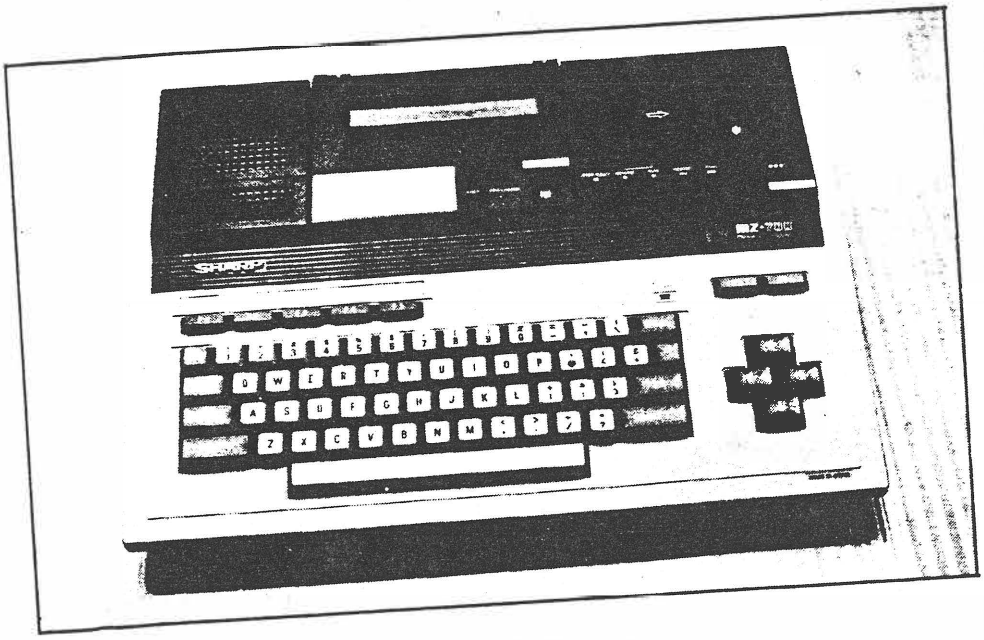
The computer comes with a separate power unit (producing 10 volts at 800 milliamps) which plugs into the rear of the machine. This is supplied with a generous three metre cable. A much shorter (around a metre) cable is provided to connect a cassette player to the VZ-200. A 'stereo' plug goes into the computer socket marked TAPE and the other end of the cable branches into two 3.5mm plugs, one each for the earphone and microphone sockets.

There are two video outlets. One connects your computer to a standard television, while the other is to drive a monitor, allowing a somewhat superior picture to be produced. Providing both these outlets is a good touch, allowing you to upgrade your picture quality if you have a monitor, without having to adapt the modulator output for it.

There are two sockets at the back of the machine which are protected by small panels, held in place by a couple of Phillips screws. They are marked 'memory expansion' and 'peripherals'. The 16k memory unit is rectangular, somewhat larger than a cigarette box, in the same pale cream plastic as the computer. The memory fitted easily into place, however, as with the Spectrum, I would not advise waving the computer around in the air with the extra memory in place.

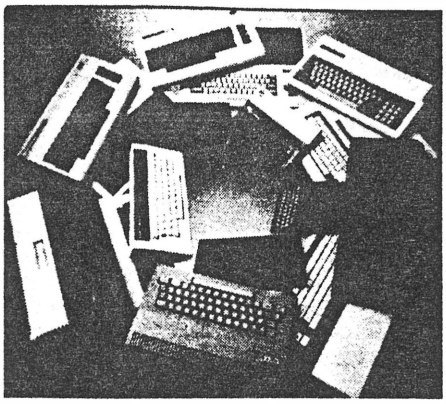
The computer comes with a hefty manual, which covers the entire VZ-200 Basic language, touching briefly (but relatively clearly, given the complexity of the subjects) on PEEK and POKE, INP and OUT (for returning the content of a port, and for sending values to an I/O port) and to USR (to call a machine language subroutine). The manual is clear, and the intention has been to make everything as clear as possible for the first-time user.

The 'Apple compatible' **CAT** is attractively designed and solidly constructed.



The Sharp's keyboard is an improvement on the Colour Genie

Oct 84 1(3) 82-87 4 of 5.



The computer/keyboard is housed in a two-toned plastic case. Most peripheral connection sockets are on the back of the unit with the exception of two located on the right hand side. Overall, the unit has a clean and uncluttered appearance.

The standard keyboard comes with eight large function keys which allow you to enter a whole command or sequence of commands with a single keystroke. In conjunction with the SHIFT and CTRL keys, up to 24 function keys can be used. Both upper and lower case letters are available in 40 or 80 column modes. The individual keys are made of tough plastic in one of three colours: light brown, bone or orange. The keyboard is ergonomically scuplted (curved) and has a very pleasant professional feel about it.

The number of potential configurations for the CAT is quite large. The following is a list of some of the components that can be added to the main unit: RS232 adaptor, communications modem, graphic plotter, 4 colour printer plotter, joystick(s), CP/M cartridge with a 48k/64k/soft emulator, cassette recorder, multiple disk drives, 128k RAM card, ROM cartridge and RGB/composite/green monitor and Super System Expander.

On the right hand side of the keyboard is a single socket for a twin set of joysticks. Each joystick has two buttons and a central control stick which unlike many other joysticks, does not return to the central position after being released.

The CAT comes with a 106 page User's Manual and a 203 page Basic Reference Manual, written in clear English and set out in a logical and orderly fashion. No index is provided in either manual.

The **MicroBee** is a very well known Australian computer. It features a real 60 key QWERTY keyboard — small but manageable.

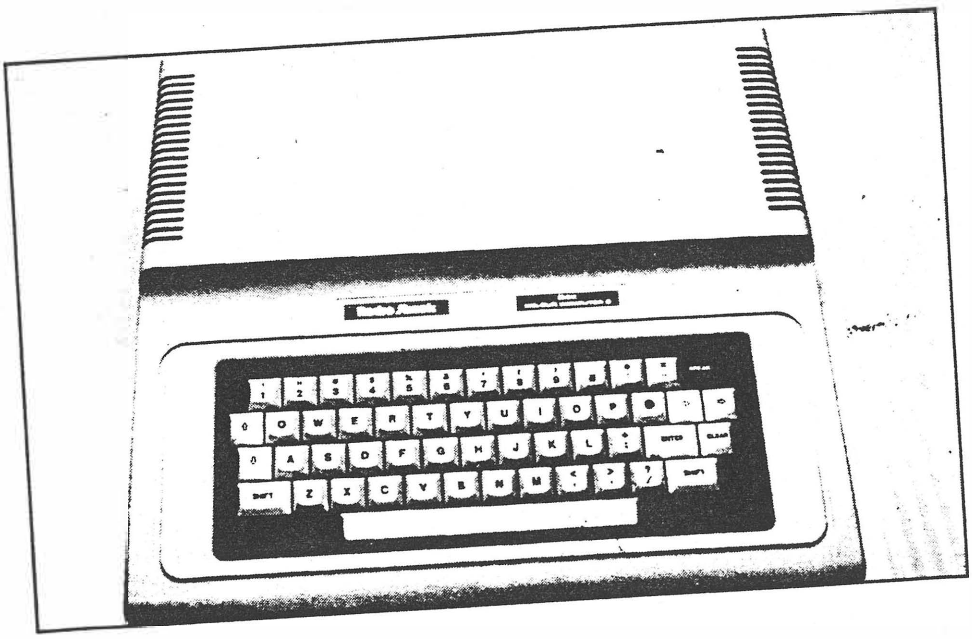
The back of the machine contains all the interface ports: power, user port, expansion interface, serial port and I/O port.

The MicroBee measures 13 inches long by 8 inches wide, 2 inches deep at the rear and a little under an inch at the front. It has a good sturdy feel about it.

The keyboard has a light feel to it with good springy keys. However, scuplted keys would help touch typists, rather than the flat surface where fingers slip off easily.

The machine will only work with a monitor (as a normal television does not have the resolution required to display the MicroBee's capability of 64 columns by 16 rows or 80 columns by 24 rows). Applied Technology, the manufacturers of MicroBee, sell monitors for $149 or can do a package deal to convert your TV.

The Basic manual supplied covers each statement, function and command in turn. The documentation is quite adequate.



The Tandy TRS-80 has a very practical look

PCG Oct 84 1(3) 82-87 5 of 5.

The snow[ball] effect

The speed at which add-ons increase in number for the home micros is nothing short of an avalanche. It is quite possible, with a little cash, to expand your Commodore 64 to a CP/M machine and to unite your 48 Spectrums in to a micro orgy. Curtis Bollington wades through all the possibilities

Now you've got what you want, you want more. You've been using your micro for a while now, you may or may not have delved into and mastered the art of programming. You've bought all the latest games (yawn!) and become a super-games-wizard-galactic-time-lord, twice. But somewhere, for some reason, something is lacking. You may have a small business, but you cannot afford to lash out on one of those IBM things, you may be wondering whether your humble home micro can be stripped of its games titles and put to work to help rake in the brass.

We here at *PC Games* like our readers to have an easy life, so this month, as part of our Home Machine Supertest we're looking at the expansion capabilities of 13 of the most popular home micros on the market, giving you some idea of how you can expand your machine. The amount of expansion varies considerably from micro to micro; it's complicated further by the number of independent suppliers of add-ons for the more popular home micros. Of course it would be impossible to list every add-on which is available for

 Nov 84 1(4) 1 of 4.

every popular home micro, although we'll mention one or two of the most popular ones.

Most manufacturers have a range of peripherals with their own label on it. This doesn't mean that it's the best one for that machine, you may find better and cheaper elsewhere by scouring the ads in *PC Games* for example.

A surprising number of computers can be upgraded to run CP/M, the operating system which enables your computer to run a wide range of business software. There are powerful word processors, financial spreadsheets, databases, and so on. In some cases you'll need to buy a second processor and extra memory before you can run these programs. Be warned, business software is a different league, gone are the $15 programs — you could find yourself paying around $600 for a software package.

Before you dash out and buy all the latest add-ons, look around at what you are trying to do, then find a software package that will perform those tasks, you may find you can get by on a single disk drive, a $50 word processor package and a $50 payroll package which will cope with about 20 staff.

Specifics

The Atari 800XL has no memory expansion although the 600XL can be expanded up to 64k. Atari have several peripherals for its machines. A disk drive is available which costs $499 for a 127k drive; it isn't possible to connect a second drive. There are two printers available bearing the Atari label, a printer/plotter which costs $159.95 and a letter quality printer costing $499.95. Atari does supply their own joysticks, but there are far better ones on the market. If you're thinking of using your Atari for business, think more of home business. Atari Writer is a word processing package costing $99.95 and there's a version of VisiCalc for $99.95. Expansion beyond this isn't available from Atari at present, but there are some mutterings about CP/M being vaguely possible in the future.

There is a disk drive available for the Sharp MZ700, a 204k single drive for $699. There are four word processors and four spreadsheets around for the Sharp, prices are approximately $40. A tractor/friction printer for the Sharp is available for $795.

There are two memory options for the Spectravideo, 16k can be added for $99 or 64k for $249. Spectravideo's 80 column printer is $549 and a green screen hi-res monitor is $199, although you may be able to pick up a cheaper monitor from an independent supplier. Spectravideo is something of a hot shot with joysticks.

The 'Quickshot' is one you may have seen in your local computer store, this retails at $19.99, there is also a version at $24.99. Disk-drives for the Spectravideo are comparatively expensive, a single 170k unit costs $999, the twin disk version has a capacity of 340k and costs $1399. CP/M is included with the disk drives but you will need an 80 column card which is a further $199.

The Sord M5 isn't a machine that you could ever want to use for a business, nevertheless there are a couple of business packages around for it. 'Falc' is a financial spreadsheet and there's a database which is based on Sord's 'Pips'. You can connect the M5 to a monitor, although Sord doesn't have its own. Disk drives are not available in Australia.

A modem is available from an independent manufacturer which will connect the M5's RS232 socket.

Add-ons for the VZ-200 are not so big. Memory can be upgraded from 8k to 24k for $79. There are no disk drives available for it which greatly hinders its capacity for small business use.

The software available for it does include a word processor priced at $29.95 and a Mailing List program priced at the normal VZ-200 software price of $12.50. Dick Smith have their own printer/plotter for the VZ-200 which costs $169 or alternatively any Centronics printer with an RS232 interface will work.

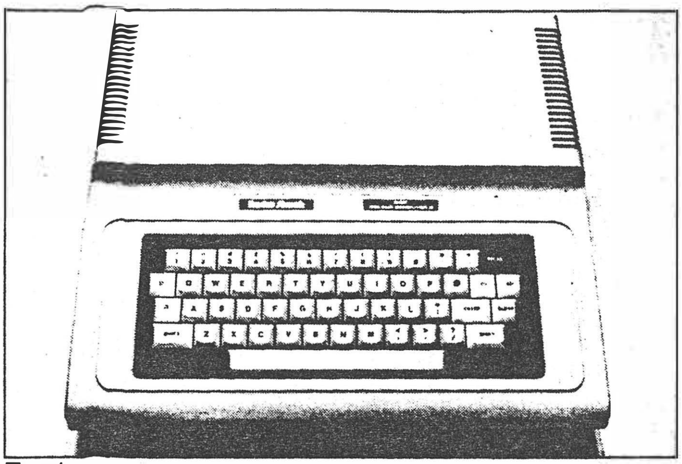
The number of potential configurations for the Dick Smith CAT is quite large. The CAT has 64k memory which can be expanded to 256k. Dick Smith have their own printer which costs $449, or any Centronics printer.

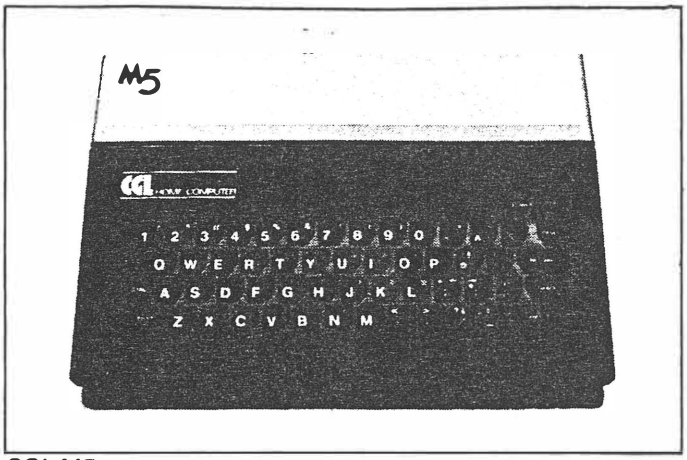
There is a range of monitors to choose from with prices ranging from $250 to $800. A Z80 cartridge with a CP/M disk is available for $395. Software available for the CAT is: VisiCalc $428, Sandy word processor $189 and Cat PFS $175.

CP/M expansion on the Commodore 64 is remarkably cheap, a cartridge costs $80. Commodore doesn't produce any memory expansion for the 64, which would make running many CP/M programs very difficult. There is a good range of business software available for the Commodore 64: Easyfile, Easyscript, Superbase 64 and The Manager. They are not particularly cheap though. A single disk drive of 170k capacity is available costing $499.

The VIC 20 and the 64 share many of the same peripherals. There is a range of printers available. The range starts with the model 801, a tractor feed only dot matrix printer costing $399. The 802

Nov 84 1(4) 2 of 4.


Tandy



CGL M5

At a glance guide to add-ons for the top thirteen micros

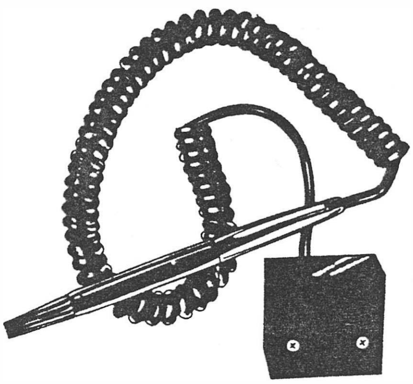
Make	Memory	Disk drives	Printer	Monitor	Modem	Networking	Joystick	Business system expansion	Software	Others
Acorn Electron	32k RAM 32k ROM Interchangeable ROM memory	100k single $358 400k single $512	Independent	12" mono $241.50 14" colour $518.00	Independent	—	With interface	Compatible with BBC business software only in disk form	View $150 View Sheet $150 Spreadsheet & WP soon avail. in ROM	50Mb hard disk 10Mb hard disk
Atari	Up to 64k max.	127k — 1 only — $499	Printer/plotter $159.95 LQ printer $499.95	Independent	Independent	—	Own, but will take any with standard i/face	—	Atari Writer $99.95 VisiCalc $99.95	—
Commodore VIC 20	Up to 16k max. $99.00	170k — single — $499	801 $399 802 $499 1101 $749	14" colour $499	Independent	Up to 8	Own $12.00	—	SimpliCalc $49.95 VIC File $59.95 Pers. Finance $39	—
Commodore 64	No options	170k — single — $499	As VIC 20	14" colour $499	As VIC 20	Up to 8	Own $12.00	CP/M Expansion cartridge $80	Easy Script $100 Easy File $80 Superbase 84 $180 The Manager $80	—
Dick Smith CAT	64k expandable to 192k	165k — single — $349	Own $449 — Any Centronics	Mono $249 Colour $389 Cal Hi-Res RGB $599 Switchable $795	Own $199.00	—	Own $34.50 pair	Z80 cartridge with CP/M disk $395.00	VisiCalc $428 Sandy $189 Cat PFS $175	—
Memotech MTX	32k $129 64k $199 128k $379	500k — single — $999 2nd drive — $599 5 Mbyte hard 10 Mbyte hard	Own dot matrix 80 column $899 Any Centronics	Independent	RS232 package incl. Comms board $149	Oxford node ring system. Up to 255 terminals $129	Independent (Atari type)	CP/M available $559	Pascal $149 New Word $189	—
MicroBee	Up to 32k $174	128k — single — $1295 twin $1595	Parallel	Own green $149.50 Own Amber $159.50 Soon to be released colour	Own $169	Starnet Beenet 1 Up to 16 terminals	Independent	CP/M 128k $1,995	Database $12.50 Spreadsheet $12.50	—
Sharp MZ700	64k max.	204k — single — $899	Tractor/friction — $795	Independent	—	—	Own $30 takes 2	—	4 word processors $40	—
Sinclair Spectrum	48k version	85k Microdrive & Interface $149	Own thermal $199	Independent	Independent	With interface one	Atari type with Interface	—	Word processors & spreadsheets are available	—
Sord	No option	Only cassette	Any Centronics	Independent	Independent	—	Own included	—	Falc (spreadsheet) database-based on 'PIPS'	—
Spectravideo	16k $99.00 64k $249	170k — single — $999 340k — twin — $1399	80 column matrix $549	Green screen $199	Own $249	—	Own Quickshot $19.99 $24.99	CP/M incl. with disk. 80 col. card needed	Disk Basic incl. in disk price	80 column card $199 Graphics tablet $179 RS232 $199
Tandy Color	64k $185	184k — single — $599 2nd drive — $499 Up to 4 drives	Any serial RS232	Independent	—	—	Own $29.95 pair	OS.9 $99.95	WP Disk $79.95 WP ROM $49.95	—
VZ-200	Up to 24k $79.00	—	Own Any Centronics with RS232 Interface $169.00	Own Colour $389	—	—	Own $39.50 pair	—	Word processor $29.95 Mailing List $12.95	—

Nov 84 1(4) 3 of 4.

model is a track or form feed 'letter quality' dot matrix printer priced at $499. The letter quality daisywheel (model 1101) is priced at $749.

The Memotech is quite something else when it comes to expansion. There is, or will be, a complete range of peripherals available ranging from three different options of memory expansion to a complete networking system allowing as many as 255 Memotechs to be used as terminals from one main terminal.

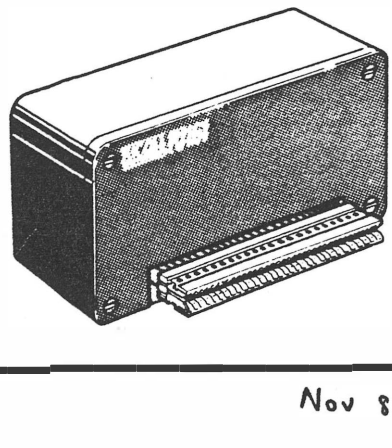
The Memotech is sold in various configurations including a CP/M system consisting of twin disks, a monitor, a printer and CP/M for $2149. Of course, all the extras can be bought separately. This CP/M system isn't the limit, if you have the money you can invest in a 10 Megabyte hard disk system, or even up to a 20 Mbyte system.

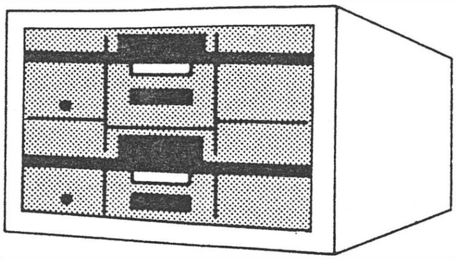


The Acorn Election comes standard with 32k RAM and 32k ROM. The ROM memory is interchangeable, and a spreadsheet and word processing will soon be available in ROM.

Disk drives are available and are essential if you want to use the BBC Model B business software. So if you want to use the business software be prepared to pay out at least $356.

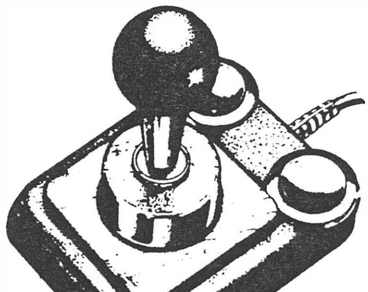
Tandy has a large range of peripherals so the Color Computer isn't short of addons. 64k of memory will cost you $185, a



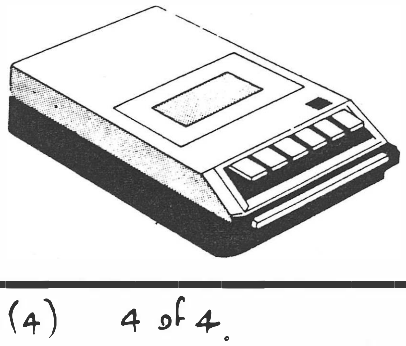


184k disk drive costs $599 and up to four drives can be added. Any printer with an RS232 interface can be used, Tandy has a range under its Tandy Radio Shack label.

The Color Computer can't be used with a monitor, only a TV. The OS-9 operating system is available for the Tandy. The software available at present are Basic and 'C'. A word processing package is available for the standard machine on either disk or ROM. The disk version costs $79.95, the ROM version $49.95.



The Spectrum uses its own ZX printer. There are two interfaces available, appropriately called Interface 1 and Interface 2. Interface 2 allows joysticks and cartridges to be used and Interface 1 is supplied with Sinclair Microdrives, which are small, fast running tape cassettes each of which hold 85k. Up to eight Microdrives can be linked together. It's



possible to expand your 16k Spectrum, but this will have to be done independently. There are many joysticks on the market for the Spectrum, it's also now possible to connect your Spectrum up to a proper disk drive. The awful rubber keyboard can also go out the window as there are some beautiful examples to replace it with which turn the Spectrum into a completely different machine. Not only does this make the machine look better, it also allows it to be used for slightly more serious applications such as word processing.

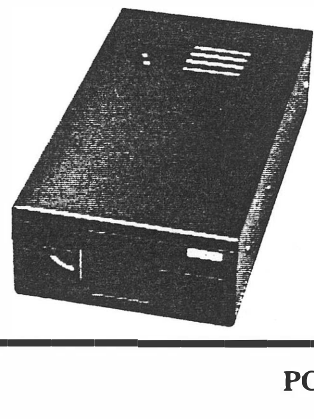
To upgrade the MicroBee to 32k costs $174. A single disk drive with 128k is priced at $1295 while the twin disk drive costs $1595.

Applied Technology's green screen monitor costs $149.50, while the amber screen monitor, also available, costs $159.50. Soon to be released by Applied Technology is a colour monitor, but as yet no price has been announced.

Networking is available with up to 16 terminals. The facility to upgrade to CP/M costs $1995 and bundled with this comes WordStar, MailMerge, Microsoft Basic, CP/M Icon Display and Multiplan.

For those not upgrading to CP/M there are database and spreadsheet programs available for $12.50 each.

The table lists most of the peripherals available from the manufacturers of the micros, it will give an indication of the extras available for the machine you own, or would like to own.



Nov 84 1(4) 4 of 4.

This was written on a 'useless' computer!

If you're wondering about the above somewhat satirical heading, it's intended to mirror the one used for "Forum" in the August issue, namely: "An important role for 'useless' small computers". How more effectively could I emphasise the validity of that article than by now using just such a computer as a word processor, to write this latest instalment?

FORUM

Conducted by Neville Williams

As you might recall, the basic theme in the August "Forum" was that small computers had come down so far in price that they could now be considered by many families as an affordable, even expendable, learning tool for the '80s.

At $99, for example, the Video Technology VZ200 (from Dick Smith Electronics) offered so much computing potential for such a modest outlay that it presented a golden opportunity for adults and children alike to gain hands-on keyboard experience — at home, in spare time, as an interesting diversion.

That the same notion had occurred to other writers and commentators was evident from the fact that it was mentioned on two or three occasions while our own article was in limbo, somewhere between the typewriter and the printing press. It has certainly been talked about since then.

As noted in the August issue, my observations were inspired, in part, by a couple of typical young families that I knew socially, in which there was evident pressure to buy a home computer of one kind or another. It is interesting to record what has happened in those homes during the intervening weeks.

Case histories

Initially, both families invested in a VZ200 basic computer, which they simply coupled to the family TV set, and both experienced a communal fascination and involvement with the games, the programs and the graphics that they were able to set up on the screen.

Objective number one — "Keyboard Konfidence" — soon became evident, with the kids variously fiddling with simple programs, practising poems on screen (even in raw BASIC), setting up "Flashwords", etc — each according to his/her age and interest.

It was about this time that father number one managed successfully to couple a portable tape recorder to his computer. Thus encouraged, he invested in a 16K memory module and, as well, obtained or contrived an elementary word processing program. From somewhere else came a printer of sorts; he was really having fun — and putting the system to tentative use as Honorary Secretary of a youth group.

Father number two was an interested observer but, over and above immediate family involvement, he had another objective in view: the ultimate purchase of a larger system for a business venture. Sooner or later, he would have to decide which to buy of those being offered to him as "absolutely and uniquely ideal" for his purpose. What he was hoping to gain was a better feeling for the whole subject.

So he bought a memory expansion module, a $40 Datasette cassette recorder and a small colour TV receiver (which the family needed, anyway) to serve as an interim monitor. He was lucky enough, also, to be able to borrow a simple printer and interface for a few weeks.

It was at that psychological moment that Dick Smith Electronics came up with a word processor program for the VZ200, on cassette for around $30. Father number two bought one immediately and set about using it for composing reports, planning documents and so on. It was consciously experimental and provisional but it allowed him to gain a much better appreciation of what he needed — and what he could afford!

As I write he has just invested in a modest but adequate business system, with a great deal more assurance than would otherwise have been possible. So, in that respect, his VZ200 has served its purpose, although I gather that he plans to leave it set up for casual use by the rest of the family.

While the foregoing might serve to validate what I was talking about in August last, the matter certainly doesn't rest there.

In that article, for example, I quoted from a review of the VZ200 in an earlier issue:

"If you want a computer to look after your share holdings, or for word processing, look elsewhere."

I didn't see fit to question that verdict because, at the time, no word processing program appeared to be available for the VZ200. There had been talk of one being written "some day" but a last-minute call to DSE brought nothing new to light.

In any case, could one take a VZ200 word processing program seriously if, as seemed likely, the text would comprise capital letters only?

Processor program

In fact, as I've indicated, a word processing program did turn up very shortly afterwards through DSE and I didn't have to spend much time with it to realise that the originators, G. Epps and M. Fackerell, had made an excellent job of it.

The program requires that the VZ200 be fitted with a 16K expansion memory module, providing a total of 24K. After loading, which takes only a couple of minutes, just over 15K of RAM is available for storing text.

Allowing an average of five characters plus one space per word, that means direct accommodation for about 2500 words of running text — sufficient for a fairly substantial essay or article, before resource to back-up cassette storage.

No less to the point, the new program enables the computer to input both upper and lower case letters to a printer so that the keyboard can be used, with Shift key, in the manner of an ordinary typewriter, The screen still displays capitals only but the text, as printed, is the normal mix of caps and lower case.

As to the VZ200 keyboard, I soon began to question, also, earlier reservations about the soft-touch "rubber" keys. In fact, they are not very different in appearance and touch from those on the Brother electronic typewriter reviewed in the August issue — and apparently enjoying ready acceptance in the marketplace.

In processor mode, the computer is completely re-programmed, with single-letter commands for most functions. Text can be typed in, then freely added to, deleted, modified, corrected, swapped around, tidied up, and so on, without any

inhibitions about lines and line numbers. It is a word processor in the true sense of the term.

After loading and pressing the Return key, the user is faced with a "menu" inviting him/her to specify what they want to do next:

(E)dit text
(C)lear text
(P)rint text
(L)oad file
(S)ave file
(V)erify file
(Q)uit program

Press "E" for Edit and text can be inserted, removed or modified, as required.

Press "C" for Clear text or "Q" for Quit the processor program and the user must verify the command with (Y)es before it is actually executed — a very desirable precaution.

Press "P" for Print, and the computer requests instructions in regard to the number of columns (20-99), single or double-spacing, left-hand margin, right-hand ragged or justified, page length and numbering, number of copies, etc.

Helpfully, each time the Menu is called up, it displays the number of spaces still left in the memory. The figure starts off at 15,042 and gradually diminishes as the stored text grows. As well, when text is being Saved on cassette, an on-screen display counts the number of characters as they are transferred.

Practical set-up

In my case, all these initial observations were made with the VZ200 system spread out on a workbench, along with

sundry instruments and tools and with an ageing EMI TV set as the monitor. I was intrigued to know how the system would appeal in more congenial surroundings as a complete budget-priced, domestic word processor — one of the roles we had originally dismissed as not worth considering!

Thinking about a monitor, I was intrigued by the possibilities of the 30cm "Princess" B&W TV receiver, which has been available for some time through chain stores like Woolworths and K-Mart. They are a good match for the VZ200 in size, colour and style and can be bought for $90 or less — complete with a 3-year warranty!

While the VZ200 program uses colour to emphasise block markers, etc, a tri-colour screen is not necessarily the best medium on which to display text. So why not a $90 monochrome monitor on which, with this program, the text would show up in white against a dark grey background?

As it turns out, the "Princess" TV receiver has a normal 50Hz mains power supply, with the internal circuitry fully isolated from the mains. This, plus a couple of video test points suggest the possibility of ultimate adaption as a video monitor. However, it worked so well with normal RF access through TV channel 1 that I did not feel necessary to pursue the matter at that stage.

What I did do was to make up a small wooden cradle on which the receiver could rest, raising it just enough (about 45mm) to allow the Memory Module and the Printer Interface to slip in underneath it. This allowed the computer to slide back against the base of the monitor, with the keyboard directly below the screen, in the approved manner!

Set up on a small (90cm × 45cm) table, with the cassette recorder on the right and the printer on the left, the system began really to look the part.

One difficulty that did arise concerned the provision of mains power. Four outlets are required, with two having to accommodate 1A plugpacks. These are too large to fit conveniently into any commercial 4-way outlet that I could find, so I made up one of my own, which I then fitted under the table for tidiness sake.

In actual use

This done, I simply sat down and "processed" the two main articles required for this issue: "Sony's Space Diversity Reception System" and "Forum". By the time I had finished "Forum", operation of the system had become almost second nature; that's how simple it is to use for running text.

There was ample room in the memory to accommodate either one of the articles, which proved handy when I wanted to flip back and add a par or modify something that I had said.

But, every now and again, I took a couple of minutes off to dump the contents of the memory on to a cassette as a precaution against a silly error, a malfunction or a power failure. As most computer operators can testify, any one of those things can wipe out hours of work in a split second and it is reassuring to have at least most of it safely on tape (or disc) as a precaution against any such eventuality.

I did, in fact, unearth one aberration in the Epps and Fackerell program: if, by accident or oversight, three block markers are placed simultaneously on the left-hand side of the screen, the memory sheds some or all of the text as rapidly as if the "(C)lear Text . . . (Y)es" instruction had been punched in! So be warned.

But, enough said!

What the excercise has served to demonstrate is that a very useful word processor for running text can be assembled around a VZ200 system and a "Princess" TV receiver for between $550 and $580 — depending on your choice of cassette recorder. It would be well suited to producing draft copies of letters, essays, papers, articles, etc, ready for final typing.

Re-inventing the wheel

At this point, some may feel that I have devoted a whole article to re-inventing the wheel — but I don't think so. It is true that, every day, countless thousands of Australians produce letters, papers and articles on word processors but the vast majority of them would cost at least four or five times as much as the small, very useful system that I've just described.

You'd prefer to produce finished rather than draft text? And tackle more elaborate work? In the main, that would involve investing in a more elaborate printer, compatible with the VZ200 — something that father number one, mentioned earlier, is currently contemplating.

FOOTNOTE: At this point in the article, calling up the menu indicates that 2705 character spaces remain unused in the memory. Subtracting that figure from 15042 gives the length of text as 12337 characters; dividing by 6 puts the number of words at 2056 (approx) — a handy check if the requirement is to produce an article of specified length.

2 of 2.

not indexed

Brisbane. "Courier Mail".

	Dick Smith VZ200 8k	Spectra -Video 32k,80k	Microbee 8k,16k,32k	Sharp MZ721 64k	Texas TI99/4A 16k	Sinclair Spectrum 16k,48k	Sord M5 20k	Canon X-07 8k	Comx 35 32k	Tandy Radio Shack 4k,16k	Atari 600 (16k), 800 (64k)	Commodore Vic 20 (5k), C'dore 64k	Sega SC-300 32k,48k	Oric -1 64k
Home TV interface	yes	yes	no	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes
Min. width on screen 40 cols	32	yes	yes	yes	32	yes	yes	20	yes	32	yes	Vic 20:22 64:yes	yes	yes
Full color graphics	yes	yes	no mono	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes
Standard type-writer keyboard	rubber keys	rubber keys	yes	yes	yes	rubber keys	rubber keys	calculator style	calculator style	4k:calc. 16k:yes	yes	yes	rubber keys	improved calculator
Music:built in thru home TV	thru TV	built in	built in	built in	thru TV	built in	built in	built in	built in	built in	built in	thru TV	thru TV	built in
Capacity for moniter	no	yes	yes	yes	no	no	yes	no	no	yes	yes	yes	yes	yes
Version of microsoft Basic	yes	yes	mic/world Basic	yes	TI Basic	yes	yes	yes	own Basic	yes	yes	Commodore Basic	yes	yes
Expandable to 64k	24k	256k	yes	yes	52k	no	32k	20k	yes	4k to 20k 16k to 32k	16k to 64k 64k to 256k	Vic 20 to 32k 64:64k	32k to 48k 48k:48k	yes
Interface with standard printer	no	no	yes	yes	no	no	yes	yes	no	4k:no 16k:yes	yes	no	yes	yes
Cartridge capacity	no	yes	no	no	yes	no	yes	no	no	4k:no 16k:yes	yes	yes	yes	no
Disc drive available now	no	yes	no	no	yes	no	no	no	no	4k:no 16k:yes	yes	yes	no	yes
Standard audio cassette port	yes	no.own tape port	yes	cassette built in	yes	yes	yes	yes	yes	yes	no.own tape port	no.own tape port	yes	yes
Price	$199	32k:$399 80k:$599	8k:$399 16k:$449 32k:$499	$499	$199 See Texas story	16k:$299 48k:$399	$350	$350	$299	4k:$179 16k:$349	16k:$399 64k:$599	Vic 20:$299 64k:$499	32k:$329 48k:$429	$399

Christmas invaders

The 'occupation' may be short-lived if you're unfamiliar with the equipment

Sales of video games consoles and home computers peak around Christmas and prices have dropped considerably over the past few months. If your kids tend to hang out in video arcades, buying a games console or a computer for Christmas may be the way to keep them at home. But if you don't know what you're buying, the equipment may turn out to be a seven day wonder.

CHOICE has tested the more afford-able models among games consoles and computers. Our findings will help you to choose the right system for you, but if you don't know a thing about com-puters, you'll be a lot wiser after reading *Time for a home computer* (CHOICE, March 1983) and the update in August 1984.

For the games that kids like see *Kid's CHOICE* on page 26.

The options

Choosing the right games equipment for your family isn't easy.

First you have to decide whether you want to buy a games console or a computer.

A games console comes complete with controls and usually a couple of games cartridges. It's all you need to play games – apart from a TV set – but after Boxing Day you're likely to be under pressure to buy more games cartridges – and they cost about $40 each.

Nevertheless, if you're simply after occasional entertainment and don't in-tend to become involved in computer technology, then a games console is what you need.

The alternative is not a computer but a computer *system*. An advertised price of less than $200 for a computer might look tempting, but once you add on the cost of all the components you need to play games the comparison is much less favourable.

Computer systems are for those who want to learn programming – either for professional reasons or as a hobby. They are also of interest to compulsive video games players who either cannot afford more and more cartridges, want to write their own games or type them in from magazines and books.

Software we bought:

(All were cartridges except where indicated)

With ATARI and CBS ColecoVision: Activision Barnstorming; Atari Air Sea Battle, Berzerk, Dodg'em, E.T., Missile Command, Pac-Man, Space Invaders

With BIT-90 and CBS ColecoVision: CBS Cosmic Avenger, Donkey Kong, Mouse Trap

With COMMODORE VIC-20: OZI-Soft Vic-20 Get Lost (cassette), UMI (Imagineering) Satellites and Meteorites

With DICK SMITH VZ 200 (all cassettes): Ghost Hunter, Invaders, Metric Spycatcher, Speed Reading*, Spellomatic*, Super Snake

With SPECTRAVIDEO SV-318 (all cassettes): Introduction to BASIC*, Armoured Assault, Spectra Home Economist*, Spectron

With TANDY TRS-80 Colour Computer: Monster Maze, Project Nebula

With TEMPEST MPT-03: Nibblemen, Alien Invader

With VECTREX: Bedlam, Clean Sweep

* not rated for entertainment value as they are not games.

What we bought

Our technical purchasing section checked what was available for under $350 and came up with four games consoles and five computer systems.

The price had to include a minimum of two software game cartridges and a joystick control even if the joystick wasn't absolutely necessary for the games supplied. In fact we often got packages which included much more.

The systems and the prices are listed in Tables 1 and 2 on pages 30 and 31.

All systems but one connect to a TV set (or a special monitor). The VECTREX has a built-in screen which means the rest of the family can still watch TV when somebody is playing.

The test

Once you have a new games console or computer system you'll be impatient to start playing, so anything difficult to set up will be frustrating. We tested how easy it was to set up each system, then played games on each one, recording the shortcomings of both hardware and the games programs.

All units passed the electrical safety tests.

Problems

Early this year we surveyed CHOICE subscribers and their children about their experiences with games consoles and computer systems. According to the survey, both are reasonably reliable, but users complain about the time it takes to get repairs after a breakdown. Repair costs are generally low – this may be because many systems are still under warranty.

We had our test samples of the CBS ColecoVision console and the expansion module for Atari cartridges replaced under warranty because the module didn't work and we were not sure whether the fault was in it or the console.

Commodore cartridges fit the VIC-20, but one cartridge from the IMAGINEERING software company for the same computer could be inserted only with difficulty.

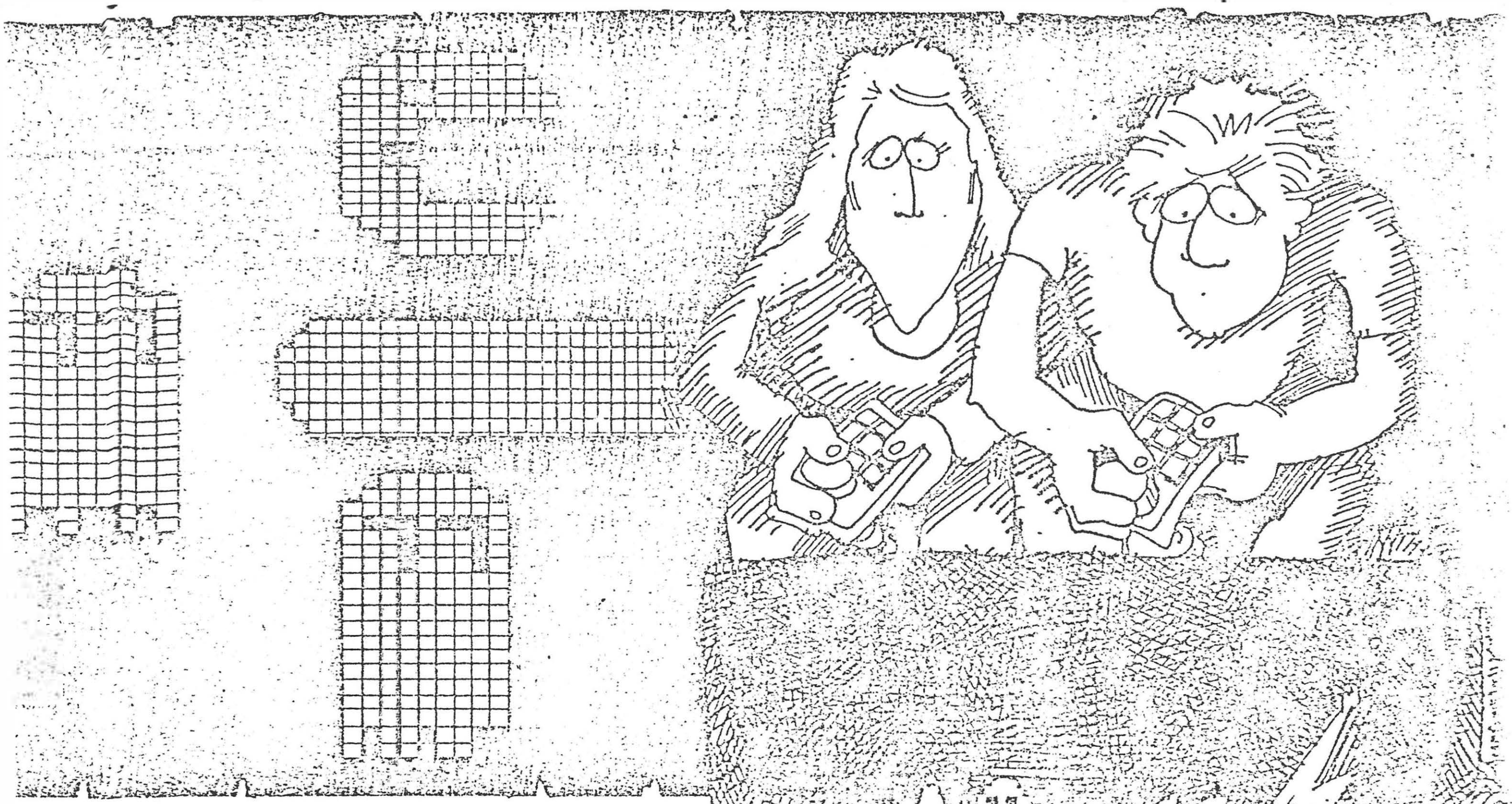
The tape disconnected in a cassette with a SPECTRAVIDEO game program.

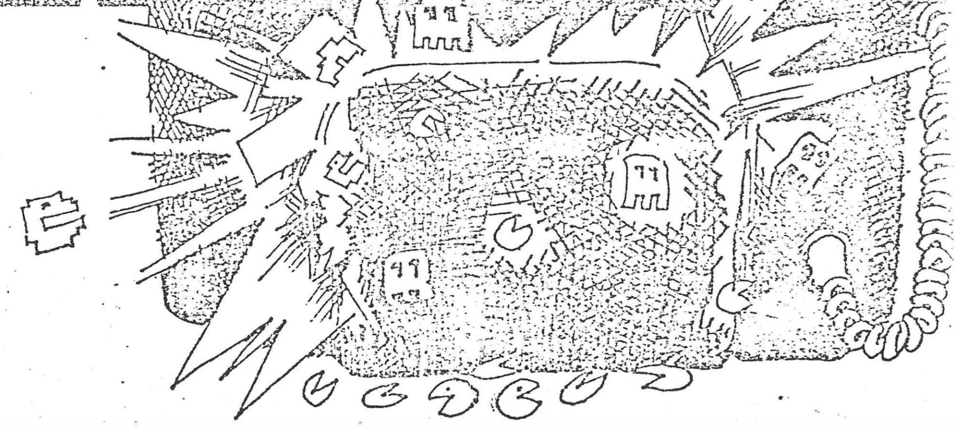
Some of the people we met in computer shops warned us that joysticks often don't last long. In fact, no joystick broke during the test but we didn't subject them to a durability test.

Connections

The games console or computer is connected to the TV through its antenna socket. Connecting and disconnecting cables every time you want to switch from TV to games system and vice versa is a bit tedious, so buy a switch-box –

1 of 4.





it lets you select the connection at the flip of a switch. Some systems come with a switch-box, others don't. You can buy one for about $4. Some new television sets have a composite video signal connection for games and computers. This connection bypasses the tuner, allowing better resolution than the antenna input and eliminating the need for a switch-box.

Buying a special video monitor is another way of improving image resolution for games and computer display. It improves the picture from a video cassette recorder, too. Video monitors are expensive, but there is sometimes a cheaper alternative – a TV set minus the tuning circuits.

Good and bad joysticks

The fun in video games comes from beating the machine. So you need a joystick which transmits hand movements directly and accurately to the screen and has the right 'feel'.

To give the right feel the joystick should be spring-loaded so that it always returns to the central neutral position.

There are two types of joystick:

- micro-switch type, 8-directional play, a good choice for beginners.
- analogue, potentiometer-type, 360° play, a bit more difficult to use, but you have more control.

Games consoles come with joysticks, but if you buy a computer it's usually an accessory you have to pay extra for. Compare the 'original' joystick with models supplied by independent manufacturers – there are many to choose from, and the original may not be what suits you best. Some 'independent' joysticks also fit the ATARI and CBS games consoles and should be considered if you need a replacement.

Lots of possibilities

The most advanced games console we tested was the CBS ColecoVision. CBS have tried to provide games as similar to those in arcades as possible, and one of the accessories is a 'racing car module' with steering wheel, gearshift and throttle pedal. It comes with the Turbo game cartridge. There is also an adapter module which we bought that allows you to use Atari cartridges. And if you find you should have bought a computer and not a games console, you can supplement CBS ColecoVision with the Adam computer module.

The BIT-90 computer can use ColecoVision games, but not the Atari or steering wheel modules.

Programs for games consoles come in cartridge form, those for computers (including games) are available as cartridges, cassettes and floppy discs, but not in all forms for all computers (see Table 2). Cassettes are cheaper than cartridges and can be used to store your own programs. Some computers can use an ordinary cassette recorder; others, like the SPECTRA-VIDEO SV-318 and COMMODORE VIC-20, need a unit especially adapted for the system. Either way, to load a program into a computer from a cassette takes a couple of minutes depending on the complexity of the program.

With cartridges the game is ready to play immediately you slot the cartridge in. You pay for this convenience, and buying overseas cheaply may not be the solution – for example, Atari cartridges from the US (for the NTSC TV system) don't work in a console made for the PAL system we have in Australia.

2 of 4.

Floppy discs, the third option for computer games and other programs, let you store lots of information in a minimum of space. The disc drive loads the computer in seconds. It too is expensive, and is not available for some of the cheaper systems.

Our survey

The notion that video games consoles are a passing fancy is not confirmed by our survey.

Some users did report they soon got bored with the games and wanted more variety and challenge. But others found exactly what they wanted in a games console. More than half (52%) said they would consider buying the same type of games console again, and brand loyalty was even higher, although CBS ColecoVision owners were generally more satisfied than ATARI owners.

Among owners of computer systems, those with a COMMODORE were more satisfied than the average – 85% would buy the same type of system again and no less than 92% would buy the same brand.

In all, 80% of owners of computer systems connected to TV sets would buy a similar system again, and no less than 85% of owners of computer systems with separate monitors said their investment was good enough to be repeated.

Not surprisingly, the more sophisticated the system, the more time is devoted to it. Owners of games consoles spent an average seven hours a week with them. Owners of TV connected computers played for eight hours a week and those who had a computer with a separate monitor spent 13 hours a week with it.

The time spent at the video game console or computer is taken from other activities, mainly TV viewing.

Unfortunately, 17% of the young respondents report they spent less time studying after they got the computer or games console. Parents who buy a computer system for a child with the objective of improving his or her scholastic performance may not get the hoped-for effect.

Computers with limitations

The computers we tested this time all have a limited capacity. To play more complex games and for professional or educational purposes you'll need an optional RAM expansion board or module. Another limitation is the number of characters per line on the screen – but some computers have an expansion module which increases the number. If you have no doubt at all that you or your youngster will be into computers for keeps it could be a good idea to look at slightly more advanced computers than the ones we tested. A more advanced model in the range may use the same

Table 1 GAMES CONSOLES (in alphabetical order)

Brand/model	Manufacturer/ distributor	Origin	Price ($) RRP/paid	Guarantee
ATARI 2600	Futuretronics	Hong Kong	199/159(1)	90 days
CBS Colecovision	CBS Electronics	Hong Kong	249/249(2)	90 days
TEMPEST MPT-03	Tempest Electronics	Hong Kong	50/ 80(3)	90 days
VECTREX	Milton Bradley	Taiwan	229/ 99	90 days

(1) package deal incl six cartridges. RRP incl one cartridge (Space Invaders).
(2) package incl Atari expansion module (RRP $75) and one cartridge.
(3) package deal incl one game cartridge

☐ Recommended

peripherals, so the difference in cost may be small – or negligible, if you have to buy expansion modules for the basic model.

On the other hand, if you have doubts whether you or the person you buy the computer for will get hooked on programming, buy something cheap so the loss will be small if the fascination wears off after a few weeks.

One of the most asked-about applications of home computers is word processing. It's out of the question with a cheap model unless is has a big memory and should only be considered with computers having a good, typewriter-like keyboard. CBS ColecoVision recently offered a complete system with the Adam computer module and a printer for $1099. That's one of the cheapest available – with a daisy-wheel printer.

We are currently testing the computing aspects of these units (and others) and will report on them next year.

Assessment

CBS ColecoVision is the only games console with expansion possibilities – it can be connected to the Adam computer and peripherals for, among other things, word processing. Even if you're not interested in expanding it, we recommend it for its good graphics and sound, the variety of software available and the quality of its joysticks.

At the price we paid, $249 including the adapter module for Atari cartridges and a Donkey Kong CBS cartridge, it's the top value in the test. The main disadvantage is the cost of more cartridges – they start at about $30.

ATARI 2600 is the most sold of the games consoles. It's extremely easy to use, the joysticks operate smoothly and there is an almost endless variety of games available, most with good entertainment value.

The special package offer we got ($159) including six games cartridges, was good value – Atari cartridges are as expensive as ColecoVision ones.

BIT-90 ranked highest as a games machine among the computer systems – but not necessarily for other uses. One of its advantages is that it accepts CBS cartridges, and with these it produced the best graphics in the test.

As with other computers you need a cassette recorder if you want to do your own programming or use software in the form of cassettes, but the whole system is inexpensive and good value.

The main disadvantage is the rubber keyboard – all rubber keyboards we tested were prone to non-keying.

Manufacture of the COMMODORE VIC-20 has recently been discontinued but it's definitely worth trying to get if you want to learn to write your own programs and plan to do a lot more than play games on your equipment.

The keyboard is its greatest asset. There's a lot of software available for it, and the system can be used with the same accessories for the Commodore 64 (which remains in production).

A minor drawback is that the VIC-20 requires a dedicated cassette recorder – but it's cheap, only about $50.

SPECTRAVIDEO SV-318 is a compromise – it has some great features but isn't perfect for either games or computing. Games are less detailed than the ones you get with CBS and ATARI cartridges, and there aren't many to choose from. But picture quality is good.

If you want to write your own programs, the computer has a comparatively large memory capacity and can be extended – but at the same time, the rubber

Table 2 — HOME COMPUTERS (in alphabetical order)

Brand/model	Manufacturer/ distributor	Origin	Price ($) RRP/paid	Guarantee	Type of keyboard	Memory RAM available	Memory RAM expandable to	Dedicated cassette recorder needed?	Input for disc drive	Input for cartridges	Input for printer	Whole TV screen used?	Interchangeable joystick plugs?
BIT-90 PAL	Electronic Warehouse	Taiwan	199/169 (1)	90 days	rubber	ns	ns	no		✓		✓	✓
COMMODORE VIC-20	Dick Smith†	England	229/179	90 days	typewriter	5K	32K	yes	✓	✓	✓		✓
DICK SMITH VZ 200	Dick Smith	Hong Kong	na/229 (2)	90 days	rubber	8K	24K	no*		✓	✓		
SPECTRAVIDEO SV-318	Rose Music	Hong Kong	na/299	90 days	rubber	32K	96K	yes	✓	✓**	✓	✓	✓
TANDY TRS-80	Tandy	Korea	250/250	90 days	typewriter	16K	64K	no	✓	✓	✓		

na not available
ns not stated
* with interface
** with adaptor

† Dick Smith has taken over remaining stock of the Vic-20, but peripherals are also available from Commodore Business Machines.

(1) package incl two joysticks and one cartridge, 32K memory
(2) package incl joystick, 16K RAM module, cassette recorder and six program cassettes.

✓ has this feature
☐ Recommended

Table 3 — PERFORMANCE
(in order of preference as a games system within the groups)

Brand/model	Keyboard action	Joystick's feel & response	Sound	Graphics	Entertainment value of software tested*	Ease of operation	Instructions	Overall ranking
GAMES CONSOLES								
CBS Colevision	●●●	●●●●●	●●●●	●●●●	●●●●●	●●●●●	●●●●	●●●●●
ATARI 2600	na	●●●●	●●●●	●●●	●●●	●●●●●	●●●●	●●●●
VECTREX 300-A1	●●●	●●●	●●	●●●●**	●●	●●●●●	●●●●	●●●●
TEMPEST MPT-03	●●●	●	●●	●	●●	●●●●●	●●●●	●●●
COMPUTER SYSTEMS								
BIT-90 PAL	●●	●●●	●●●●	●●●●	●●●●†	●●●●	●●●●	●●●●
COMMODORE VIC-20	●●●●	●●	●●	●●●	●●●●	●●●	●●●‡	●●●●
SPECTRAVIDEO SV-318	●●	●●●	●●●	●●●●	●●●	●●●	●●●●	●●●●
DICK SMITH VZ-200	●●	●●	●●	●●●	●●	●●●	●●●●	●●●
TANDY TRS-80	●●●●●	●	●●●	●●	●	●●●●	●●●●	●●●

* what you buy may be quite different
** monochrome screen
† with CBS cartridges
‡ good handbook but downrated because games instructions come on loose sheets

the more dots the better
☐ Recommended

keyboard is a disadvantage. This system needs a special tape recorder if you use software on cassettes. The SPECTRAVIDEO is worth considering – a good buy if you can forgive the keyboard.

VECTREX 300-A1 has been discontinued, which explains the sale price of only $99 – less than half the recommended retail price. The VECTREX has its own high resolution picture screen, so you can play without interfering with the TV viewing of other family members. However, there are very few games available for it and the ones we tested were very similar. Cartridges may not be available in the future, so if you buy this unit, it would be wise to pick up a good supply of games cartridges at the same time.

DICK SMITH VZ 200 is not a good games machine. Games for it are not very detailed, picture quality is smeary and the joysticks are poorly designed and cannot be replaced with one from another manufacturer. However, it is a suitable computer for those who want to learn BASIC programming despite the rubber keyboard, which tires the operator and occasionally doesn't register when a key is pressed. It's inexpensive and it works.

TANDY TRS-80 Colour Computer is another system that's more suitable for computing than for games.

The games we tested were rather boring, picture quality was poor and smeary, and the range of games available for the TRS-80 is small. The joysticks are flimsy and don't return to centre by themselves. If you want to use it for computing it's worth thinking about – it had the best keyboard response in the test.

TEMPEST MPT-03 is a very basic games console with only two levels of skill and unexciting software. Its worst point is the poor design of the joysticks – its best the simplicity of operation.

It's cheap but you run the risk of it being put away on a shelf within a few days – and if that happens, it's not a good investment.

What to buy

This rating is based on performance for video games use only, not computer function. There is one exception to the order of preference from the test – VECTREX has only been rated acceptable because it's no longer manufactured and there's great uncertainty about future service and software supply. The situation is different with the also discontinued COMMODORE VIC-20 – there is a well established market for VIC-20 software, and it uses the same peripherals as the Commodore 64, which remains in production.

RECOMMENDED (in order of performance – video games use only)

	Price ($*)
CBS ColecoVision (g)	249
ATARI 2600 (g)	98
BIT-90 PAL (c)	199
COMMODORE VIC-20 (c)	149
SPECTRAVIDEO SV-318 (c)	299

ACCEPTABLE

VECTREX 3000-A1 (g)	99
DICK SMITH VZ 200 (c)	99
TANDY TRS-80 Colour Computer (c)	300
TEMPEST MPT-03 (g)	50

*pre-publication price check
c computer
g games console

4 of 4

Back to the VZ-200

From a reader in Oak Flats on the NSW South Coast comes a letter which is set out in the accompanying panel. I suggest you read it at this point.

In responding to W.T's letter, I have a strong urge to do so in similar terms: "Whoa! Slow down there."

For sure, I made a case, in the August '84 issue, for investing $99 on a VZ200 computer — a product that had been dubbed by some buffs as "useless". I did so on the basis that, for $99, it could offer members of a family a unique opportunity to gain hands-on experience and, with it, a degree of confidence, when faced with a larger computer at work or at school. I quoted examples of how this had already occurred in typical family situations.

Out of all this came the further notion of using the VZ200 as the basis of an inexpensive word processor — something for which there was an obvious opening. It worked out better than ever expected, helped along by a $90 "Princess" B&W TV set as a monitor, a 16K memory module, a mini-printer and interface, a cassette recorder, and a word processor program that had fortuitously become available on tape from DSE. The exercise culminated in "Forum" for November '84 entitled: "This was written on a 'useless' small computer".

I might add that, since then, many more such articles have been written on that same small word processor and on other systems like it. The pity of it is that, as I write, supplies of the VZ200 are in danger of drying up, just when their bargain price utility has become most apparent.

Far from disproving anything that I have said, W.T's letter carries the idea of $99 self-tuitional exercise well beyond anything that I had really considered. He gives no information as to his educational background but, if to begin with, he was as much a computer novice as he makes out, he has had his $99 worth several times over!

I'm not about to debate his remarks about the ultimate accuracy of the VZ200, because I certainly haven't devoted to it that kind of attention. Nor do I propose to. I'll happily leave that to other readers who may share W.T's enthusiasm for such exercises. In the meantime, someone who should know was not the least surprised by his observations.

Computers, he said, work to certain limits of accuracy, determined by their logic resources and speed of processing. Like most other products, they are designed with a market role and price in view. If user needs dictate a higher order of accuracy than a certain computer will give, the buyer's only option is to purchase a better one. As it is, the ultimate accuracy of the VZ200 is quite typical for budget priced PCs.

But while W.T. pursues further enlightenment on that score, I'm more impressed by the apparent build-up in the skills and potential of this hitherto unemployed reader. He should, by all means, keep probing and asking questions but, in the meantime:

Good on yer, mate!

from Neville Williams "Forum"

Forty years ago it cost a fortune. Can we do it now for $99?

Dear Sir,

Whoa! Slow down there with the eulogies to the VZ200 $99 computer. The monitor in ROM has a bug in it.

I followed with interest your praise of the VZ200 in "Forum". Even on the dole, $99 isn't too hard to scrape together so, when I saw the DSE advert in July, a trip to DSE in Wollongong became mandatory.

After acquiring rudimentary programming skills, I hit upon the idea of making my self-education more interesting by repeating Mauchly and Eckert with ENIAC, in calculating e (or pi) to a large number of decimal places (pi to 2040 places, "Scientific American", Dec '49, p. 30).

To begin, I wrote a program to print the product of any two integers, however large, exactly and was rewarded with intermittently correct results. Mostly it was correct but occasionally (the frequency increased as the computer warmed up) incorrect answers were outputted.(!)

After running the same thing on the demonstration CAT at Wollongong, to make sure it wasn't a bug in my program, I remembered an early exercise that had caused the VZ200 to crash:

10N = 1:INPUT S:FOR P=1 *to* S:N=N*P/(P+1):?N:: NEXT: RUN

If one RUNS *and then* INPUTS *23 two times, the second time the computer goes crazy.*

I had been informed that it was only POKEing into a memory location it didn't like and didn't think was important!

As a consequence, the VZ200 pays for itself many times over in the self-education required to debug the machine language monitor. In the meantime, it is not possible to use the computer for any calculations requiring great accuracy. Even the double precision feature available by using the STR$ and VAL functions is inconsistent in its output.

Is any other VZ200 user out there able to help me?

I don't hold anything against DSE but it would be nice to say that any Tom, Dick or Harry can do in 1985 with a $99 what the computer buffs did in the '40s and '50s with computers costing a fortune.

By the way, the Tandy "Understanding" series books are okay and they're cheap!

W.T. (Oak Flats, NSW).

Dick Smith's new VZ-300: THE BABY SURE HAS GROWN!

Jim Rowe

Following its very successful VZ-200 'baby' personal computer, Dick Smith Electronics has just released an improved version called the VZ-300. It has also announced a new low-priced floppy disk system, to go with either model. So for a really penetrating review of these new products, we passed them over to someone who was pretty deeply involved in the development of the original VZ-200 . . .

I HAVE TO ADMIT that I was really quite keen to check out the new VZ-300 personal computer. During my years at Dick Smith Electronics, one of the projects I spent quite some time on was the development and support of the little VZ-200. I believed then, and I still believe now, that the VZ-200 turned out to be an excellent 'first computer' for beginners — cheap, yet surprisingly powerful. Obviously quite a few other people thought so too, because DSE has apparently sold over 30,000 of them.

Perhaps my enthusiasm for the VZ-200 might seem to make me biased, but I don't think so. While on the whole I believe the VZ-200 turned out well, it certainly wasn't perfect. Like every other model on the market it had its shortcomings, and as someone who worked on the project right from the beginning I've probably had more insight into these than most.

Right at the outset, I should say that overall I'm very impressed with the new VZ-300. It is very much better than the VZ-200 in a number of ways, and certainly a worthy successor to it. Considering that DSE is selling it for the same price as the initial price of the VZ-200 — $199 — that makes it even better value for money.

That said, there are a few disappointments. Earlier shortcomings which still haven't been fixed, the odd irritating new one, and areas of incompatibility with the earlier model (some of which were probably unavoidable). Luckily most of these are relatively minor. But let's look at the positive side first.

Improvements

The most obvious improvement over the old VZ-200 is the keyboard. In place of the original array of rather rubbery tablets (the Yanks call them "Chiclets" after the US brand of chewing gum), the VZ-300 sports a much more professional full-size moving key array in the standard typewriter configuration. There's now a normal space bar at the front centre, and two shift keys in the normal positions. These are very big improvements, making the new model much more suitable for word processing. Great!

The case of the VZ-300 is a little bigger than that of its predecessor: 305 x 183 x 56 mm compared with 290 x 163 x 51 mm. It is also made from slightly darker plastic — much the same colour as the IBM-PC. It not only looks better, but is also provided with better ventilation so that it runs cooler.

The other main improvement isn't obvious until you start using it. The new VZ-300 has considerably more inbuilt random access memory to store user programs and their data. This is distinct from the 'video RAM', used to store the information displayed on the video monitor or TV screen; both the new and old models have 2K of video RAM.

Instead of the 6K bytes of user RAM provided in the original VZ-200, the new model sports a full 16K — nearly three times as much. This is a very worthwhile increase, and means that many users won't need to worry about extra RAM.

Of course there is extra RAM available, in the form of plug-in cartridges as there was for the VZ-200. In fact there are now two RAM cartridges, one to provide a further 16K bytes and the other described as providing 64K.

Another improvement, albeit relatively minor, is that the VZ-300 is fitted with a small switch underneath to disable the colour part of the video signal. This means that if you are using the computer with a monochrome video monitor or TV set which is incapable of displaying colour, you can switch it off to clean up the display.

The VZ-200 was fairly irritating in this respect, with a constantly moving Moire interference pattern on the screen. The main cause of the pattern was a beat between the 3.58 MHz clock signal used for the computer itself, and the 4.43 MHz signal used for the video colour subcarrier. Early VZ-200s were particularly effected, but later machines used a reverse video format (ie, dark lettering on a bright screen) and improved internal shielding, which made quite a difference.

The new VZ-300 still has the reverse video format, and also has a completely reworked main circuit board inside — so the shielding may be further improved. The DSE catalogue blurb suggests that the main system clock frequency has been shifted from 3.58 MHz to 3.54 MHz, although I haven't had a chance to check this. If this is so, it was presumably done to reduce the Moire problem.

One way or another there does seem to be less pattern evident on the screen, although it is still there and mildly irritating even with the colour switched off.

By the way, the DSE catalogue suggests that the VZ-300 has additional colour display capabilities compared with the earlier model. This doesn't seem to be evident from the user manual, and some quick tests certainly didn't show up any extra display modes. So if there are any, they're well hidden.

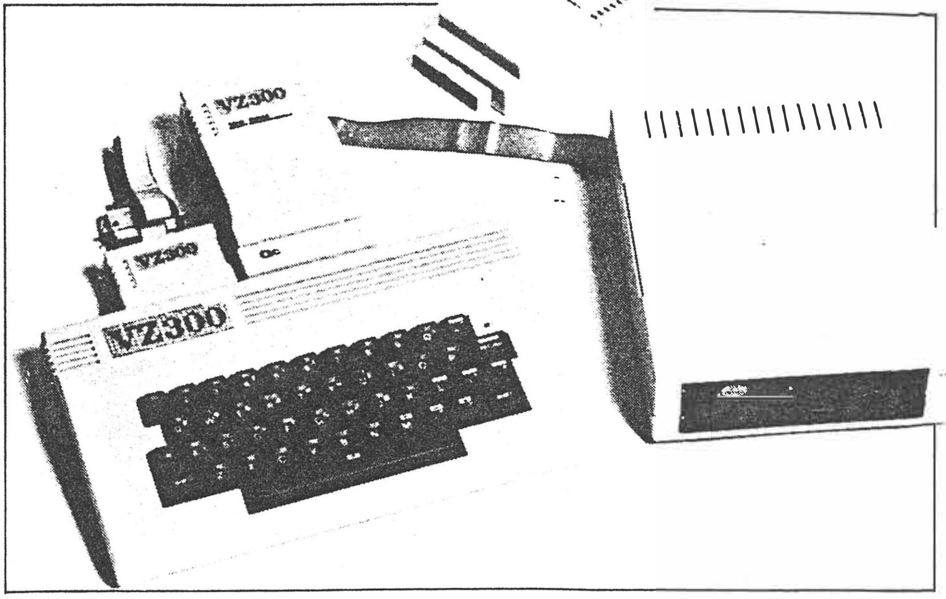
Like the later versions of the VZ-200, you can swing between the 'green characters on black' and 'black characters on

1 of 5.

TABLE 1. BASIC VZ-300 SPECIFICATION

Processor/speed	Z80/3.5 MHz
Internal User RAM	16K
Internal ROM	16K
Keyboard	46 keys, typewriter format
Video format, text	32 x 16
Graphics	64 x 32, 128 x 64
Colours	8/9
Inbuilt I/O	video, VHF, cassette
Cassette data rate	600 baud
Power supply	12 V/1 A (adaptor supplied)

Expansion capabilities:
- 16K RAM expansion cartridge
- 64K RAM expansion cartridge (see text)
- Twin joysticks with interface
- Centronics-type printer interface
- Data cassette recorder
- Floppy disk drive with power adaptor
- Disk controller cartridge
- Four-colour printer plotter



green' modes for text and lo-res graphics if you wish, by using POKE statements (POKE 30744,0 and POKE 30744,1). Doing this in a program in conjunction with the COLOR statement effectively gives you another pair of background colours, and one more character colour: black.

Could be more

Now for the disappointments. I suppose the first of these is the one already noted, that the Moire problem is still evident. But I recall that this problem was a particularly difficult one to solve, so perhaps we should be tolerant here.

Frankly I was more disappointed to find that the internal BASIC in ROM is unchanged from that in the later VZ-200s. It is still a partly nobbled version of Microsoft Level II, with useful things like ON GOTO, ON GOSUB, DEL, STRING$, TRON, TROFF, AUTO, VARPTR, DEFINT, DEFDBL, DEFSNG, DEFSTR, and double precision maths still all disabled. Since the BASIC is fully licensed from Microsoft, I know of no reason why these functions could not have been activated for the VZ-300. It would have made it much more powerful, even more powerful than the original TRS-80 and System 80 for only one quarter the price. What a pity this wasn't done.

Other disappointments come to light when we look at the VZ-300's RAM expansion cartridges. And it's here that things start to get a little complicated.

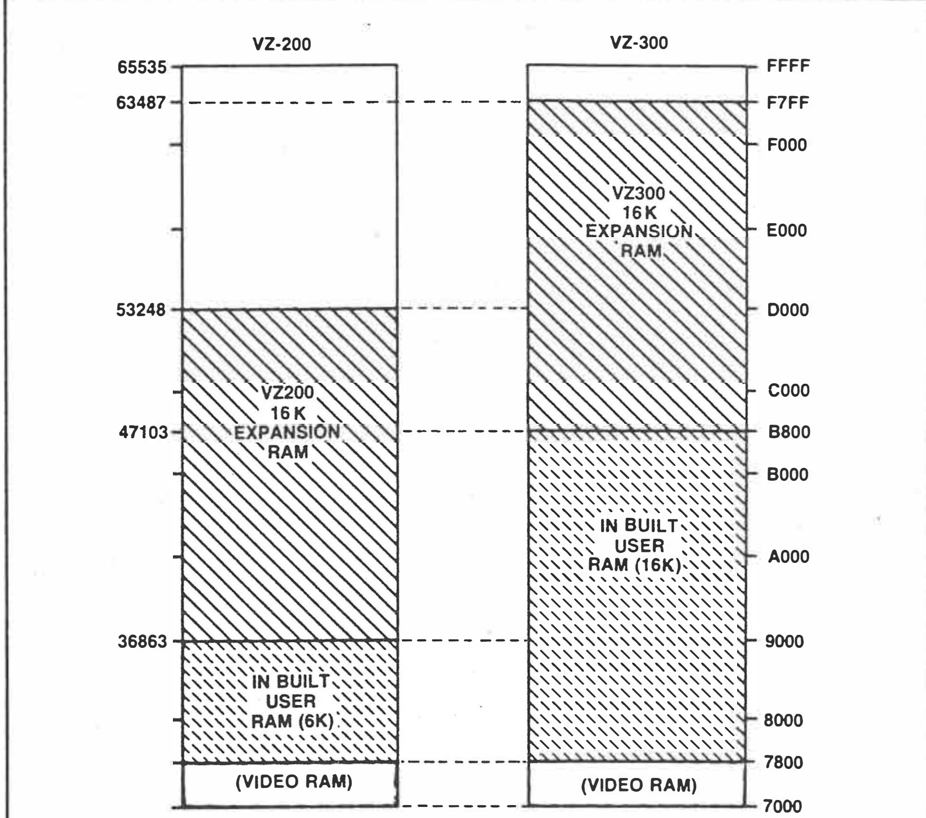


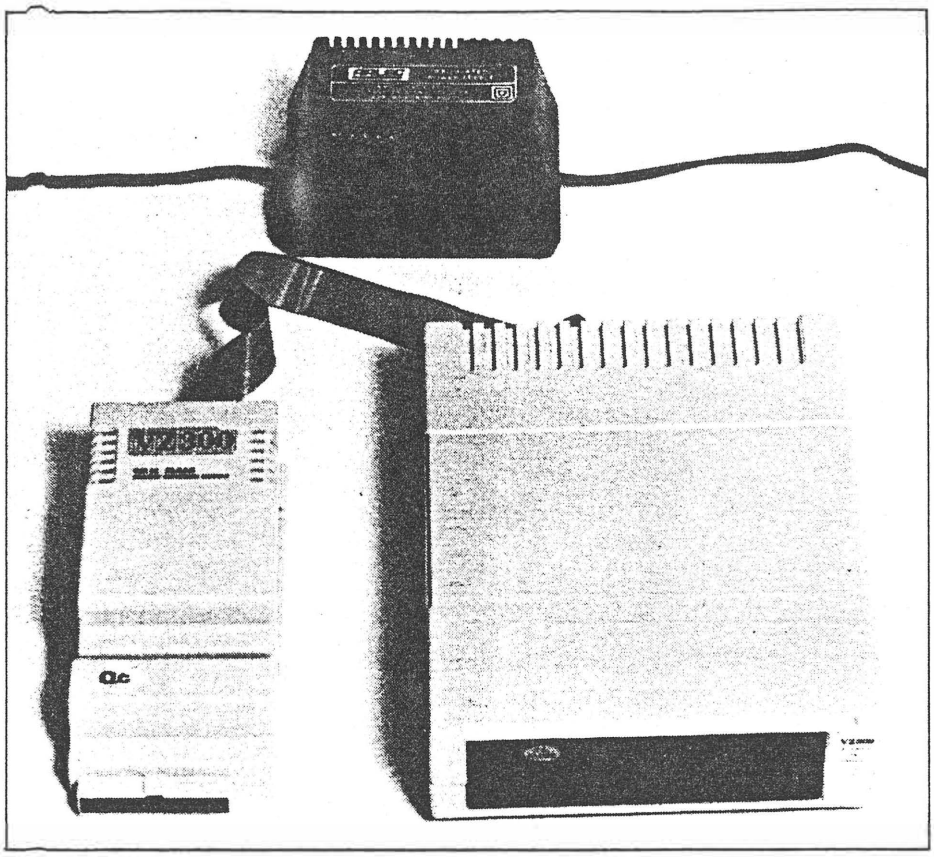
Figure 1. The memory maps for the VZ-200 and VZ-300. Note how their internal and expansion RAMs cover different address ranges.

First there's the matter of compatibility with the VZ-200. In its latest catalogue, DSE says that both modules will also work with the VZ-200. While it's true that they'll both plug into the VZ-200, this is really quite misleading — particularly for the 16K cartridge.

With the original VZ-200, the internal 6K of user RAM extends to address 8FFF hexadecimal, or 36863 decimal. The VZ-200's 16K expansion cartridge provides as you'd expect 16K of extra RAM, starting at 9000 hex or 36864 decimal and extending to CFFF hex or 53247 decimal.

However because the new VZ-300 has 16K of internal user RAM, the internal memory already extends up to B7FF hex, or 47103 decimal. So naturally the VZ-300's 16K expansion cartridge starts at B800 hex or 47104, and extends up to F7FF hex or 63487 decimal — only 2K short of the top of memory space. This means that the two 16K memory expansion cartridges cover different address ranges, making them at least partially incompatible (see Figure 1).

If you plug the VZ-300 cartridge into the older model it will function electrically, but the BASIC interpreter won't be able to use it. In fact it won't even know the extra memory is present, because there will be a 10K 'hole' of unoccupied memory addresses ▶



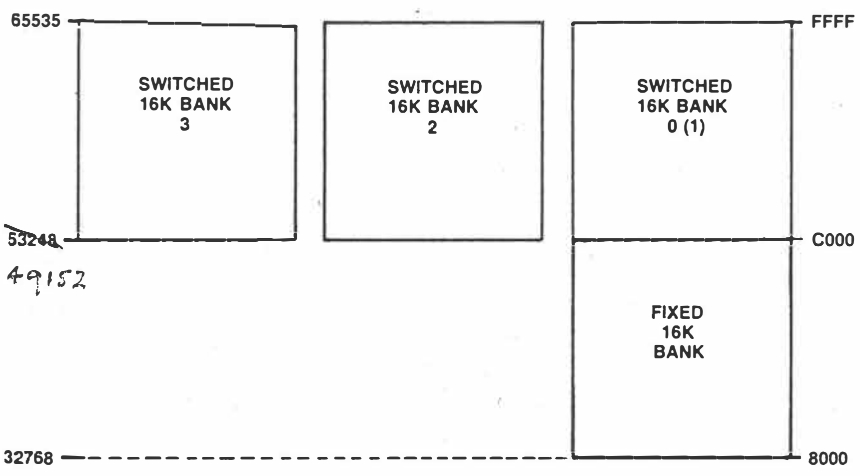


Figure 2. The new '64K' RAM expansion cartridge uses software bank switching to provide three alternative banks for the top 16K of memory space.

(9000 — B7FF hex) between the top of the internal RAM and the start of the expansion RAM.

When the VZ-200 powers up, its operating system checks how much RAM memory is fitted by running up the addresses with a quick write/read test. As soon as the test fails, it calls the address of the last successful test the 'top of RAM'; in other words, it tests for the top of *contiguous* RAM.

So if you try this out, as I did, you find that the VZ-200 completely ignores the extra RAM and makes no use of it. Which is just as well, because the 10K chasm in memory space could cause all sorts of crashes and weird software problems!

The VZ-200's 16K expansion cartridge won't work properly with the VZ-300 either, although in this case it does give *some* extra RAM — not 16K, but a measly 6K. Again Figure 1 shows why: the only additional addresses it provides are from B800 to CFFF hex, or 47104 to 53247 decimal.

In a way this is a bit of a pity, because people with the original VZ-200 won't really be able to make full use of their old 16K cartridge if they 'trade up' to a new VZ-300. If they use it, they'll still only get a total of 22K of user RAM — exactly the same as they had before (ie, 24K overall counting the 2K video RAM).

Of course this really arises from the fact that the VZ-300 already has an extra 10K of internal RAM, occupying the extra memory addresses. I guess it's one of the prices you pay for having an improved model with much more RAM in it already!

It would have been nice if the original 16K cartridge had been fitted with a switch, for changing its memory addresses to suit either model. Why didn't we all think of that at the time? (Alright, nobody's perfect!)

But to summarize, the old and new 16K RAM expansion cartridges are NOT interchangeable. Each is really only suitable for use with its own model — although you *may* be able to use the old one with the new computer if you don't mind getting only 6K of extra RAM.

When it comes to the '64K' cartridge, there isn't so much a compatibility problem as one of functionality.

Because of the way the VZ-200/VZ-300 memory space is allocated, with user RAM starting at address 7800 hex or 30720 decimal, both models can only have a total of 34K bytes of user RAM effectively functional at any instant. So the designers of the computer had a problem when it came to providing a '64K' expansion cartridge.

They solved it by using a technique known as "bank switching". The 64K of available RAM is divided into four 16K chunks or banks, one of which is arranged to permanently occupy addresses 8000 to BFFF hex (32768 — 49151 decimal); this largely overlaps the existing internal RAM. The other three banks are all arranged to occupy the remaining 16K of addresses, from C000 to FFFF hex (49152 - 65535 decimal); see Figure 2.

Of course there isn't much point in having all three banks simply working in parallel, so a pair of flip-flops at a special address (7F hex) in I/O (input-output) space is used to switch only one of the three banks on at any particular time, under software control. By writing a code number to this I/O address, a program can switch from one bank to another. The code numbers for the three banks are 0 (or 1), 2 and 3 respectively.

So although the whole 64K can't be written to or read from at any particular instant, programs can turn the banks on and off. Or to be more exact, *machine language* programs can do this. BASIC programs can't, because the VZ-200/VZ-300 BASIC interpreter keeps its stack and string variable buffer at the top of available RAM. So if a BASIC program tried to switch memory banks, vital information would be whisked

 3 of 5.

away from the interpreter, and the system would 'crash'.

In other words, only machine language programs can take advantage of the extra 32K of RAM available in the 64K cartridge. With BASIC programs, the cartridge can effectively only be used as a 32K cartridge. This applies with both the VZ-200 and the new VZ-300.

There is a difference, though, because of the way the 64K cartridge's RAM starts at 8000 hex and overlaps the internal RAMs. With the VZ-200, you get an additional 28K bytes over the basic machine. Whereas with the VZ-300 you only get an additional 18K, a mere 2K more than you get with the new 16K cartridge.

So for BASIC programmers (probably the vast majority) the 64K cartridge is really only worthwhile for the VZ-200. With the new VZ-300 it only gives you 2K more than the 16K cartridge. Worth remembering, when you consider that it's nearly double the price!

The only other mildly disappointing thing about the VZ-300 is the user manual. Instead of the three separate original manuals, all user material has now been jammed into a single overstuffed comb binding. No doubt this saves a few cents, but it also makes the manual very much harder to open flat for use. It's one of those silly little things that could easily have been avoided.

Despite all of these little disappointments and irritations, the new VZ-300 is still a very good little computer. Hence my comment earlier that I believe it's even better value for money than the VZ-200. In fact it must surely be the cheapest possible way to get a complete colour computer, suitable not only for learning the fundamentals, but then for being expanded and put to practical use.

By the way, the other VZ expansion items all seem to work just as happily with the new VZ-300 as they did with the earlier model. This includes the Centronics printer interface, 4-colour printer/plotter, joysticks and data cassette recorder. As far as I can see there are no compatibility problems with these at all.

Disk drive and controller

Talking of expansion, this leads me to the other new release from DSE, the VZ disk drive and controller. Here again the news is good not only for buyers of the new VZ-300, but for owners of the VZ-200 as well; because the new disk system does indeed seem to work equally well with both models. And it brings a whole new order of operating convenience and efficiency to both.

The basic disk system consists of three items of hardware: the controller cartridge, the disk drive itself, and a power supply adaptor for the disk drive.

The controller cartridge plugs into the rear of the computer, into the same connector normally used by the expansion RAM cartridges. However, so that you can still use a RAM expansion cartridge with the disk controller fitted, it has a further connector on the top to receive the RAM cartridge. It's quite a neat arrangement.

On the back of the disk controller cartridge are two 20-way sockets, each of which can receive the ribbon cable from a disk drive. In other words, the controller is designed to handle not just one, but two drives if you wish. The sockets are marked "D1" and "D2", and naturally enough if you have only one drive, its cable plugs into the D1 socket.

The disk drive is a compact half-height 5¼-inch unit, in a moulded plastic case which matches the VZ-300 and the controller cartridge cases. The ribbon cable leading to the controller cartridge is permanently attached to the drive case. The only other connection is a 5-pin DIN socket which takes the power for the drive, from an in-line type power adaptor. Each drive needs its own adaptor, while the power for the controller cartridge comes from the computer supply.

So much for the hardware for the disk system, which is quite neat and straightforward. Now for the interesting part: how it works. The manual and brochures are very sketchy about this, but after a bit of detective work and checking it out with a few test routines, I think I've worked out the basics.

As far as I can discover, the disk drives and controller use a simplified storage encoding system something like that used in the Apple II computer family. There doesn't seem to be a dedicated disk controller chip in the controller cartridge, just an 8K byte ROM and a few housekeeping chips. And the disk drive electronics is simpler than for the usual SA-400 type, with only a few basic signals conveyed each way along the cable to the controller. For example the drive has no opto-detector for the disk index holes, so there is no index signal.

So far so good, of course. The simple disk system used in the Apple II family has proved a particularly reliable one over the years, and if the VZ system is similar then it too could well turn out to be just as reliable. And the lack of a detector for the disk index holes means that like the Apple disk system, the VZ system can use either soft or hard sectored disks equally well. I tried this out in fact, and both types of disk worked beautifully. Great!

DOS

By now, the more experienced readers are no doubt starting to ask "OK, OK, but what about the DOS?" (For the not-so-experienced, a DOS is a disk operating system, or the program needed to look after all of the housekeeping jobs involved in storing information on the disk, and then retrieving it again.)

Glad you asked. Inside the controller's 8K ROM, along with the machine language routines used to control the disk drive itself, there looks to be quite a tidy little DOS — or more accurately, a little disk BASIC. In other words, a set of routines which patch themselves into the existing VZ ROM BASIC, to provide it with the extra BASIC commands to cope with basic disk operations. You get these disk BASIC commands as soon as you turn on the computer with the disk controller plugged in; they don't have to be loaded into RAM from a system disk.

The controller's 8K ROM doesn't gobble up valuable memory addresses normally used by RAM, either. It occupies a range of otherwise vacant addresses down below the RAM area, between the top of the BASIC ROMs at 4000 hex (16384 decimal), and the VZ's keyboard array at 6800 hex (26624 decimal). So when the disk system is installed, you still have as much RAM as before. It's very neat and efficient.

Now if you're an experinced old pro or hacker looking for a really fancy bells-and-whistles DOS, forget it. VZ disk BASIC has a pretty modest set of commands. But on the other hand if you're a newcomer who's never used a disk system before, it has all the disk commands you're likely to need for a long, long time. And they're nice and simple to use, as they should be.

The commands are listed in Table 2. As you can see, they provide all of the basic things needed for preparing disks, loading and saving both BASIC and machine language programs, maintaining disks, checking disk status and doing simple sequential data storage from BASIC programs.

How does the VZ disk system check out? Not bad at all; in fact considering what it is designed to do, it does it particularly well.

First of all, I tried formatting a few blank disks using the INIT command. It took about 75 seconds per disk, which compares quite well with most other disk systems. Then I tried loading in a few decent-sized BASIC programs from cassette tape, saving them on disk and re-loading them, to compare these disk operations with doing the same things via tape. That's the ultimate test.

The results were fine. Take for example a program of a little over 6K, which took about 82 seconds to save to tape and another 82 seconds — after the start of the program had been found — to verify or load again. With the disk system this program took only about 12 seconds to SAVE (including an automatic verify), and only 7.5 seconds to LOAD again. So the disk system ▶

4 of 5.

is about 14 times faster than tape for saving, and about 11 times faster for loading. And very much more convenient, of course.

By the way, the VZ disk system uses a fairly standard single density storage format with 40 tracks each of sixteen 128-byte sectors. This gives 624 sectors, or 78K bytes of formatted storage per disk. Not enormous, but quite practical.

I tried out just about all of the disk commands and functions, which all seemed to operate very reliably. In fact it all worked without a hitch of any kind, not only with the new VZ-300 but with my son's original model VZ-200 as well.

Of course the more experienced user will tend to be a little disappointed at the lack of some of the fancier DOS functions like those for random access (PUT, GET, FIELD, MKD$/I$/S$ and CVD/I/S etc). But that's not really relevant here. This system was designed for the typical user, who mainly wants to load and save programs quickly and easily. It does that, and it does it well.

All in all, I'm quite impressed with the VZ disk system. Of course compared with the basic VZ-300 it's not cheap; the disk drive and its power adaptor alone will cost you $249, more than the computer itself. And you still need the controller cartridge, at $79 more. But it's still very modest compared with the cost of other disk systems.

So there you have it. A new and improved VZ-300 computer, and a beaut little disk drive system for both models. Despite a few minor disappointments, they're both really good products.

TABLE 2. VZ DISK SYSTEM — COMMANDS

INIT	Formats a blank diskette for use (either soft or hard sector)
DIR	Lists the files on a disk
STATUS	Gives available storage space on disk (in both sectors and bytes)
SAVE"filename"	Saves BASIC program to disk with filename given (8 chars maximum)
LOAD"filename"	Loads named program into memory without executing
RUN"filename"	Loads named program and starts execution
REN"oldname","newname"	Rename disk file
ERA"filename"	Erase disk file
DRIVE n	Change currently used disk drive (n = 1 or 2)
BSAVE"filename",s,e	Save binary file (eg, machine language program), with filename given, starting at address s and ending at address e (both in hex)
BLOAD"filename"	Load named binary file into memory
BRUN"filename"	Load named machine language program into memory and begin execution
DCOPY"filename"	Copy named disk file from one disk to another
OPEN"filename"	Open a data file for write or read
PR#"filename"	Write data to opened disk file
IN#"filename"	Read data from opened disk file
CLOSE"filename"	Close disk file

5 of 5.

An Electronics Australia review

DSE's new VZ300: word processing for the masses

With stocks of the popular $99 VZ200 personal computer now virtually exhausted, DSE has announced a substantially upgraded replacement, model VZ300. It will have its own special appeal to computer enthusiasts but, as well, it opens up a whole range of options as the basis of a relatively inexpensive word processing system.

by NEVILLE WILLIAMS

My first encounter with the original VZ200 was when I took one along on a holiday and, rather than overdo the relaxation bit, I coupled it to a TV set in the flat and worked my way through the manuals. In the process, I realised its potential tuitional value, which became even more apparent when the original $199 purchase price was later reduced to $99.

Subsequently DSE came up with an excellent cassette-based word processing program, written for the VZ200 by Messrs Epps and Fackerell. On screen, it provided means to compose text in takes of up to 15,042 characters, and to freely correct, delete, insert or shuffle words, phases or paragraphs, rearrange copy, etc, using simple, easy to remember commands.

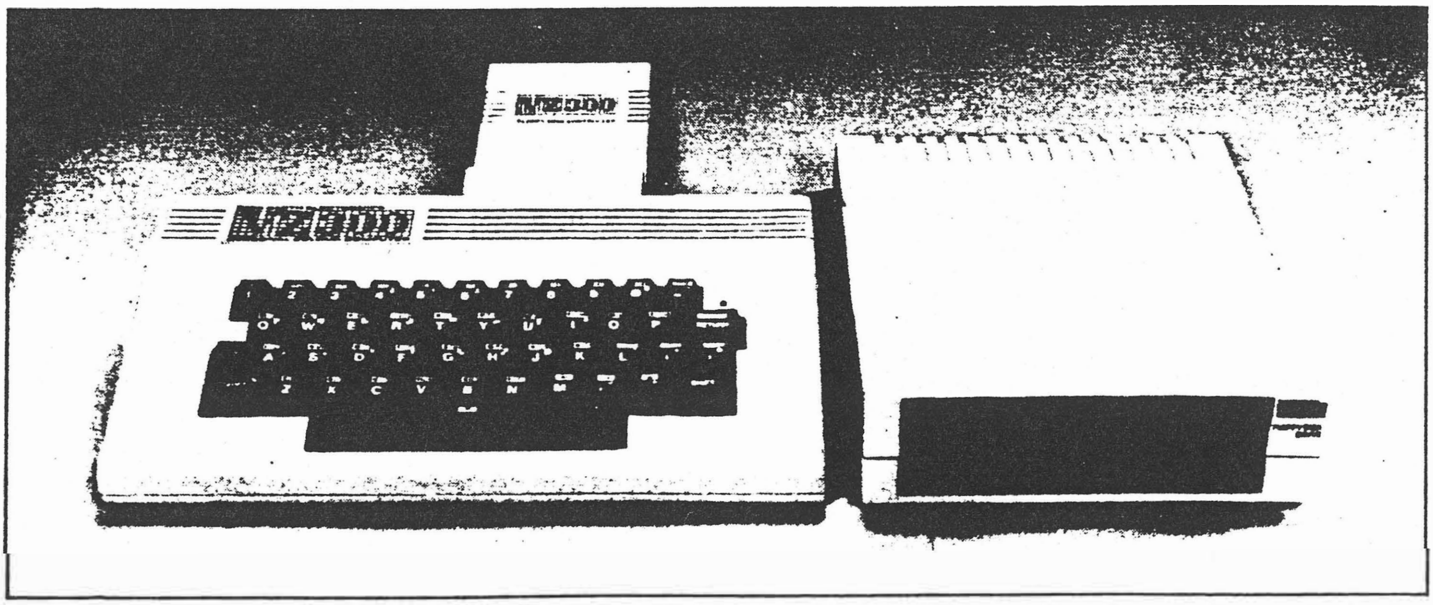
The copy could be stored on cassette tape or fed to a printer as a normal mix of capital and lower case letters, numerals, symbols and punctuation marks. There was provision to specify the length and width of print, left and right margins, indents, columns, right-hand justification, etc.

It added up to a modest but practical word processor for about $550 all up, and still under $1000 with a more pretentious printer. (See "Forum" for November '84.)

The exercise served to introduce quite a few people to the advantages of word processing and to whet their appetite for something more ambitious — an option which the new VZ300 opens up. But, first, we summarise what it offers as a basic personal computer.

While quite obviously developed from the earlier model, the VZ300 is somewhat larger overall at 305(W) × 183(D) × 63(H)mm. It is housed in a moulded plastic case, grey-green in colour, with peripherals to match.

Like the VZ200, it has an on-off switch at the right-hand end, and sockets at the rear for a plug-pack power supply, for video out and RF out (TV channels 0-1) and for cassette tape in-out. Also at the rear are ports for a floppy disk controller and/or optional expansion



The VZ300 Computer with the DOS (disk operating system) cartridge plugged in at the rear. It, in turn, has a "piggyback" socket for RAM or ROM modules. On the right is the VZ300 Floppy Disk Drive.

1 of 5.

memory, etc, and a Centronics type printer interface.

The most obvious difference is the keyboard, which now has proper keys and a normal space bar, instead of the flat "rubber" pads fitted to the earlier model. They certainly look more professional and lend themselves to a higher typing speed. In action and "feel", the keyboard is much the same as found in other modestly priced PCs.

Inevitably, perhaps, the larger keys have crowded out the "Function" legends which appeared below the pads on the earlier model. The functions are still active and accessed by the same keys but now need to be memorised, or identified with the aid of a separate card. In practice, they are not used all that much.

A further omission is the colour coding above the numeral keys but this could presumably be corrected in due course with a suitable adhesive label.

Accessible through the bottom of the housing is a small colour/B&W slide switch — a welcome provision, when used with a monochrome monitor. With the original VZ200, the 3.58MHz clock signal could in some cases, produce a noticeable interference pattern.

It was usually not troublesome on a receiver/monitor because of the limited passband of the RF link, but it could be objectionable on a wideband monitor, unless attenuated by a 3.5MHz low pass filter in the video line.

Provision of the colour disable switch and a claimed small shift in the clock frequency appears to have considerably reduced the problem.

Internally, the layout has been completely revised to accommodate everything on a single board, with due attention to ventilation and to minimising possible hot spots in the circuitry.

A notable improvement is a substantial increase in in-built user RAM (random access memory) — from 6K for the VZ200 to 16K in the new model. This should be adequate for many purposes but external memory expansion modules in the VZ300 range of options can at least double this — an observation which calls for further explanation.

Compatible or not?

From the viewpoint of compatibility, the good news is that the Microsoft Basic II ROM is essentially the same in both models, so that software for (and from) the VZ200 should work with the VZ300 — and it does, to the extent that we have been able to verify. The printer/plotter, Centronics printer interface, cassette recorder and joysticks for the VZ200 also appear to be compatible.

The same cannot be said, however, for the memory expansion modules, mainly because of the manufacturer's decision to provide more internal user RAM in the new model. It has meant that the top address for the internal RAM (therefore the starting address in the matching extension unit) is nominally 10,000 higher in the case of the VZ300 than it is for the older model.

If the VZ200 extension unit is plugged into the VZ300, it will function but will provide only the same total memory space as for the VZ200: 22K. This comes about because it uses the same starting address in both models, simply overlapping the upper 10K of the VZ300.

It still means, however, that if you have the opportunity to trade up to the new keyboard, you can plug in the old 16K expension memory and carry right on — until you can spare $69 for the right one and the extra 10K of memory.

With its own 16K expansion module, the VZ300 provides a nominal 32K of user RAM. It is important to note, however, that the new VZ300 module will not work at all in the older model. Because of the 10K gap between the finishing and starting addresses, the VZ200 won't even know that the module is aboard!

A 64K expansion module is also available but at $149 is debatable value. The point behind this is that the BASIC Interpreter in both models (VZ200 and VZ300) can only cope directly with 34K of RAM so that, for normal BASIC programming, only 34K of RAM can be effective — so the 64k module gives a potential increase of 2K for $70!

In machine language, additional 16K banks in the 64K module can be independently selected by programming, but the facility is not available in BASIC. Curiously, the 64K cartridge would probably offer better value if used in conjunction with the VZ200, providing the same 34K of RAM — a significant increase over the previously available 6K or 22K.

While final stocks of the VZ200 were cleared at a quite low figure, the fact remains that, two years ago or more, it was hailed as a "breakthrough" at $199 for such a powerful small computer.

Now, despite rising costs, the VZ300 comes in at that same figure, with a much superior keyboard, more than double the amount of user RAM, other refinements and provision for a wider range of expansion peripherals, including a completely new disk drive and controller, described later in the article.

That must surely add up to a very attractive proposition for budget conscious PC enthusiasts.

Format conversion tape

To assist those who have accumulated cassette files compiled with the E&F word processor program, DSE have prepared a conversion cassette allowing them to be changed to the new ROM format.

The conversion tape is fed into the VZ300 (or VZ200) with extension RAM in ordinary BASIC configuration, using CLoad. When RUN, it readies the computer to receive and re-format the E&F file and displays the relevant instructions on the screen.

When the E&F file has been loaded and duly processed (the text is not displayed) it can be Saved on cassette, and can then be fed directly into a ROM format word processor, where it can be displayed, checked and re-edited if necessary.

As a word processor

With the release of the VZ300, it should be possible for anyone who has been using a basic word processing system, as mentioned earlier, simply to substitute the improved keyboard and carry right on.

In fact, by way of verification, this portion of the article is being prepared on just such a system: VZ300, an existing 16K expansion unit, DSE data cassette recorder, E&F (Epps and Fackerell) W/P program, printer interface and printer, and a "Princess" B&W TV receiver. It works well!

If setting up such a system for the first time it would, of course, be logical to purchase peripherals to suit the VZ300, partly in the interest of styling and colour, but also to ensure a full 32K of memory is available for possible future requirements.

Certain points are worth noting, however, in seeking to plan ahead for word processing facilities.

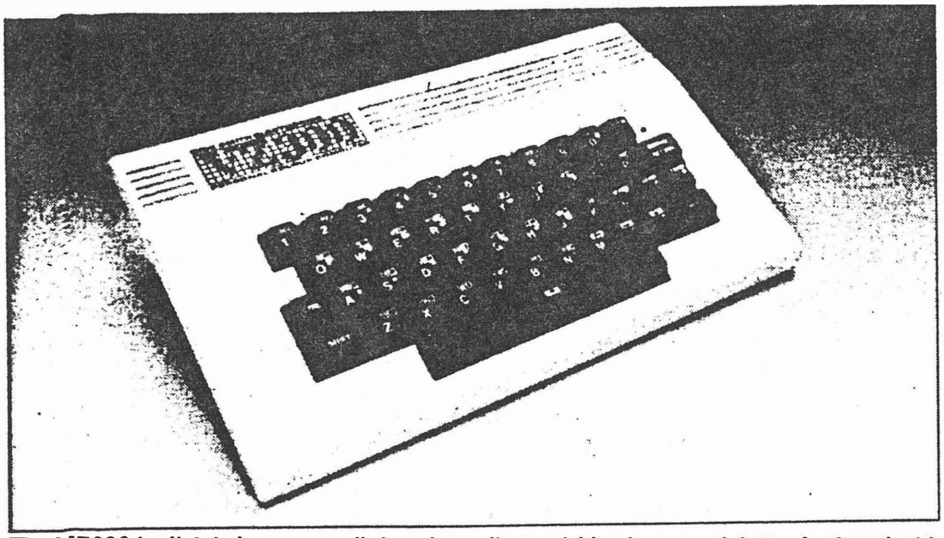
1. The VZ300, as is, will load the E&F program with memory space to spare but it will not work correctly by reason of certain "bugs". As with the VZ200, a 16K expansion module is essential.

2. The E&F program was written specifically for the VZ200 and is limited internally to 15,042 characters at a time — about the length of a 3-page article in this magazine. In its present form, it will not take advantage of the extra memory space.

3. The E&F program currently makes no provision to talk to the new DSE floppy disk memory store. If planning to buy a disk system, it will be necessary to

2 of 5.

DSE's new VZ300



The VZ300 is slightly larger overall than the earlier model but has a much better keyboard with normal space bar.

select an appropriate word processing program, such as the one that is now available on ROM (read only memory) pack.

In planning a replacement word processor program, DSE decided that it should be on ROM rather than on tape, to avoid the 90-second routine of having to load it on each occasion prior to use. However, instead of adapting the existing E&F program, they had a completely new one prepared by Messrs Dubois and McNamara, identified as the VZ300 Word Processor.

It is mounted in a plastic case similar to that used for the extension memory and plugs into the same socket. At switch-on, the Command menu appears on the screen with the options: Edit, Print, Clear Text, Disk Commands and Tape Commands. As such, it is ready for immediate use.

Fairly obviously, with the ROM occupying the extension socket, text can be stored only in the computer's internal RAM. This presents no problem in the VZ300, which can accommodate 15,564 characters at any one time — marginally more than the 15,042 available with the E&F program.

The ROM is also functionally compatible with the original VZ200 but, because of its limited (6K) internal RAM, only 5324 bytes can be accommodated at once. Except for correspondence and short articles, the user would be heavily dependent on tape or disk storage.

The new VZ300 word processor has more on-screen edit provisions than the E&F program and, at first glance, might appear to be more difficult to memorise and to use. Perhaps it is, but not by all that much — especially if one makes up a simple guide card, as illustrated.

As with most such programs, the newcomer is well advised to concentrate initially on facilities which they prefer or need to use and to assimilate the remainder only as necessary. Personally, after having used the E&F program for some time, I found no difficulty in adapting to the new one.

On a monochrome monitor, the characters normally appear dark against a lighter background, with capital letters reversed. Up to twelve 32-character lines can be accommodated on the screen at a time, with operating mode information along the top, as appropriate.

For composing and editing text, the program provides the usual facilities to move the cursor to any desired point on the screen or in the text. Alternative edit modes are available by pressing Control (9): Mode A which allows errors to be simultaneously over-typed and obliterated; Mode B, which allows characters or text to be deleted or inserted, the rest of the text being shuffled automatically to accommodate the changes.

For major insertions — new text or from disk or tape files — the cursor can be placed at the desired point and the display flipped to Insert mode by using Control(0). The new copy can then be composed, displayed and checked out on an otherwise blank screen and will be inserted at the designated point in the main text upon return to Edit mode.

A block marker is available to designate blocks of text to be moved, copied or deleted, while there is also provision to search for and change designated "strings" (words, etc) up to 16 characters long. While all this is going on an FM (Free Memory) display indicates how much memory space is still available at any time.

Of note also is the provision for TAB stops, which can be set and cancelled as required, with their positions indicated at top and bottom of the screen. The most obvious single use is to provide an Inset at the beginning of each new paragraph, obtained simply by typing Control(T).

As indicated earlier, the main Command Menu has provision to Clear Text (with a Yes/No precaution) and other separate sub-menus to do with Print, Disk and Cassette.

Cassette commands

Of these, the cassette facility is the least complicated. It provides for: (1) Save; (2) Load; (3) Merge; (4) Verify; (5) Return to main menu. The Load function calls for special comment, in that it replaces existing text in the memory and is therefore protected by a Y/N query. To add to existing text, as in Insert mode, the Merge command must be used.

The Verify command provides means to ensure that text has actually been saved but I missed the character count that is provided on the E&F program. Neither program has provision for directly cueing the cassette deck, which must be switched manually to the required function.

The VZ300 disk storage system is completely new and has the added advantage of being compatible with the older VZ200, thereby significantly increasing its potential. At around $330 all-up, disk is admittedly more expensive

VZ300 — BASIC SPECIFICATIONS

Processor/speed	Z-80/3.54MHz
Internal ROM	16K
Internal user RAM	16K
Keyboard	46 keys, typewriter format
Text format	32 cols, 16 lines
Graphics format	64 × 32, 128 × 64
Colours	8/9
Input/output (in-built)	video, RF (TV 1&2), cassette
Cassette data rate	600 baud
Power pack (supplied)	12V/1A (nominal)

 3 of 5.

DSE's new VZ300

to install and operate than a cassette system, and somewhat more accident prone for the newcomer, but it can save and load files in less than a tenth of the time it takes with cassette.

Three items of hardware are involved: a disk controller cartridge containing the DOS (Disk Operating System), the disk drive unit itself, and a dual power supply adapter for the drive unit, providing 5V and 12V at 0.7A.

The controller cartridge is designed to plug into the expansion socket on the back of the VZ300 (or VZ200) and draws its supply from the computer. In turn, it carries a "piggyback" expansion socket, which can accept the module which would otherwise be displaced — typically a 16K RAM or, in the present context, the new VZ300 word processor ROM.

The twin pack is only about one centimetre taller than the computer itself and could typically slide out of sight under the monitor.

At the rear of the controller cartridge are two 20-pin sockets, marked D1 and D2, each capable of accommodating a cable and plug connection from a disk drive. I only had one drive unit available (normally plugged into D1) but the system can accommodate two, if desired, each with its own separate power supply.

The drive unit, colour matched to the VZ300, measures a modest 190(W) × 70(H) × 260(D)mm and, apart from the disk "door" at the front, has no user knobs or switches. It is entirely software controlled from the computer, the details depending on the program in use, viz: BASIC or Word Processor.

I used it with standard 5-inch single-sided, soft sectored disks but I gather that it works quite happily with the hard sectored variety. The signal storage format is 40 tracks, each with sixteen 128-byte sectors. This works out at 624 sectors for a total storage of 78K bytes per single sided disk.

My observations with the disk store were primarily in the context of word processing and, as such, it gave no hint of bother. I simply connected it up, as per instructions, inserted a disk, switched on and waited expectantly but in vain for any reaction. None came until I pressed (D) in the main menu, for Disk Commands. Then it happened as per the user manual: strange noises and a red indicator light, indicating that it was poised for action!

The on-screen disk menu provides for:

INITIATE: Formatting a new disk with information relating to the word processor program.

DIRECTORY: A list of the files on the disk and the number of tracks available for further storage (up to 39 tracks at 2K each.

SAVE TEXT: A file name is called for, comprising up to eight characters, the first of which must be a letter of the alphabet.

LOAD TEXT: Subject to Y/N query. Use of the load function will replace text already in memory.

MERGE TEXT: Used to transfer text from disk file to a designated point in memory, without destroying it.

KILL TEXT FILE: Used to delete unwanted individual files from a disk.

RETURN TO MAIN MENU.

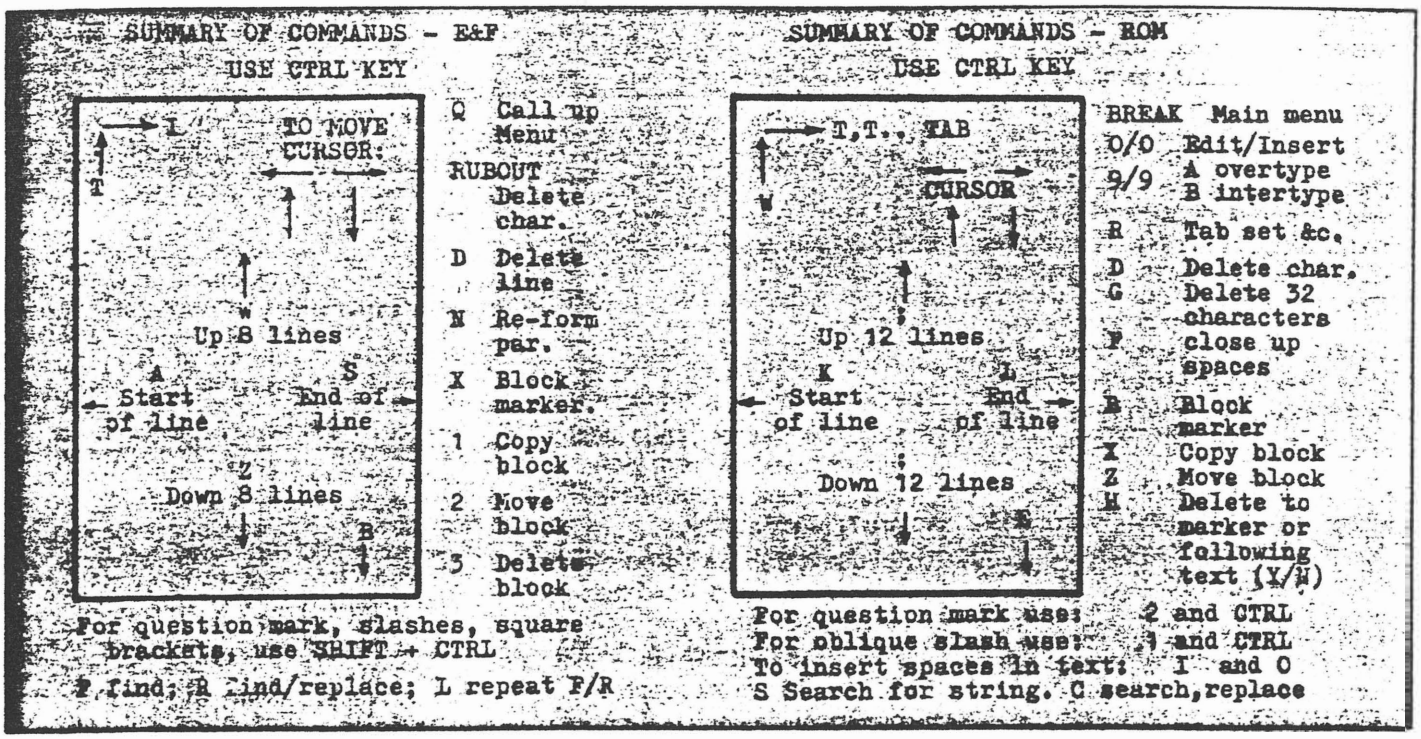
As with most new facilities, it may take a while for the newcomer to become confident with disk storage but the relative simplicity of the VZ300 system and the above menu, should ensure a head start.

How to use and organise disk facilities to advantage is probably best worked out in the light of individual needs and experience. Accepted wisdom is ultimately to install twin disk drives so that working files can be transferred to back-up disks as a precaution against accidental loss, and for long term storage.

For anyone just graduating from cassette facilities, it would probably make sense to use the single disk system as a working store, transferring completed files to cassette for long-term (and inexpensive) storage.

Print facility

Unlike Tape and Disk, the Print facility provided by the new VZ300 Word Processor does not use a separate



Typical home-made prompt cards for the E&F program (left) and the new ROM program (right). They contain most of the commands used for composing on-screen text and are helpful both for learners and for anyone needing to use more than one program.

 4 of 5.

menu, even though it involves a dozen or so potential control instructions for a typical, unpretentious printer. Unless the user has a good memory, he/she will probably have to rely on a prompt card to avoid errors and omissions.

Print instructions, preceded by a print marker, must be typed on to the screen ahead of the relevant text, giving directions as to page numbering, page length, margins, indent, justification, centring, page feed, line feed, line spacing, etc.

If desired, modified instructions can be included at points in the text, between pages or paragraphs, to change any of the relevant parameters — instructions that could be assembled and inserted very conveniently, using the Insert/Edit facility.

One of the options — D=Send to Printer (Y/N) — allows the text to be processed and inspected on screen, with selected portions being either printed or not printed, as desired.

It is also possible to print over-long documents direct from tape file or disk file, using the computer memory as a buffer.

EXPANSION OPTIONS

16K memory expansion RAM
64K memory expansion RAM
Twin joysticks & interface
Data cassette recorder
Disk drive & power adaptor
Disk controller cartridge
Centronics type printer interface
Printers, as required
Word processor ROM (see text)
Word processor cassette
Format conversion tape
Assorted software

Last but not least, a command N= . . . allows numbers to be sent direct to a suitably responsive printer, to control a variety of possible parameters to do with print face, line spacing, etc. The ultimate usefulness of this provision will, of course, depend on the printer selected.

Instructions relating to the new ROM-based word processor are in the course of preparation, being available only in draft form when we were putting the system through its paces. Indeed, in the process, we were able to make a number of hopefully constructive suggestions.

But, to sum up, if you're looking for a personal computer that doesn't cost the proverbial "arm and a leg", with word processing options that fit the same description, DSE's new VZ300 warrants close consideration.

HOW TO BUY A MICRO

How to use the buyers guide to help you choose a computer system for your home or business.

Deciding to buy a micro is the easy part, deciding which micro to buy is where your problems start. Micro computers vary in price from $100 for the cheapest home machine to over $20,000 for top-range business machines, so there is plenty of opportunity to waste your hard-earned cash.

The first step in choosing a computer is to state as clearly as possible why you want one. Be honest, as you could make an expensive mistake if you are not.

Computers can be divided into two types, though the dividing line is somewhat ambiguous. The main difference between home and business micros is in the medium used to store data or programs for future use. Home micros use the humble audio cassette, and you will actually have to buy a cheap cassette recorder for this purpose. Though the medium is cheap and readily available it is a very slow method of accessing data; witness the fact that it can take more than five minutes to load your favourite game from cassette into your micro. Business computers use floppy disks to store data, and the disk drive is usually built into the computer. These are more expensive but allow far faster access to data anywhere on the disk.

Computing

Home computers form the cheaper group, generally costing under $1000. By far the majority of home micro owners use their computers to play games, available in cassette form for around $20 to $25, but there are other valid uses! Computers are finding their way into schools more and more these days, but unfortunately most educational software is of a very poor standard.

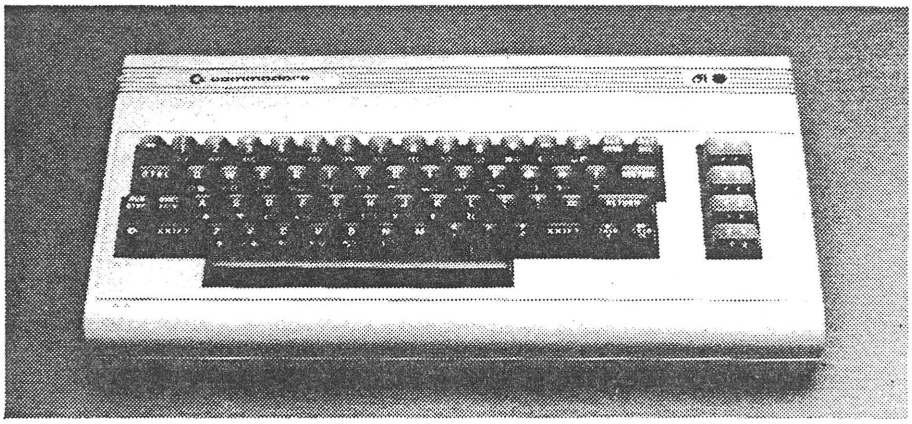
Another good reason to buy a home computer is simply to learn about computing, with a view to learning how to write your own programs for fun or profit.

However there are many misconceptions about using your computer at home. It is usually far quicker to do your home finances with old-fashioned pen and paper than on a micro, and you will find it far easier to use a cookery book than try and store your favourite recipes on it.

Business micros start at around $1500 and extend upwards, though there are a few cheaper models around. The majority of business machines come complete with one or two floppy disk drives and a monochrome monitor, which is one reason why they tend to be more expensive than home machines! Business micros come into their own in applications that require speedy access and manipulation of large amounts of data. They can replace filing cabinets, keep your accounts, deal with large mail-outs and the like. Indeed spreadsheets databases, accounts packages and word processors make up a large part of the software available for business micros. However there are plenty of more specialised applications, and programs exist to cater for the needs of newsagents, builders, farmers and many others.

However, many of the more powerful home machines can be expanded into quite respectable business systems by purchasing add-on disk drives, monitors and printers. Quite a few small businesses make good use of, say, the Commodore 64 to keep track of stock or accounts, though it is arguable that it may have been more cost-effective if they had bought a business micro in the first place.

So the answer is to take account of your future requirements as well as your immediate needs. Finally, beware of promised 'super' machines that the manufacturers haven't quite got round to making yet. It is tempting to wait for the next generation of micros, but remember that there will always be a new wonder machine round the corner, and when it does come out the chances are it won't be half as wonderful as they promised! It is better to go for something that has a good track record, and plenty of software readily available.



Home Micros

The specifications to look for when choosing a home micro depend very much on your intended use.

If you are looking for a games machine then your primary concern has to be the quantity and quality of software available for that particular model. Games written for one model will not usually work on another.

If you are looking for a home machine that is easily expandable then you should look at the type of interfaces supplied. These are the various standard plugs and sockets needed to allow computers to 'talk' to other components. Centronics is a one-way interface allowing the computer to output data only. Most printers use the Centronics interface, so if you intend to get a printer make sure the computer you buy has a Centronics port. The RS232 interface is more complex, as it is two-way, allowing the computer to receive information from the outside world as well as talking to it. It can be used with printers, but you usually have to pay extra for the privilege. RS232 is primarily used to connect one computer to another, so they can communicate and share data.

Certain manufacturers, most notably Commodore have their own standard

Dec/Jan 86 3(1)

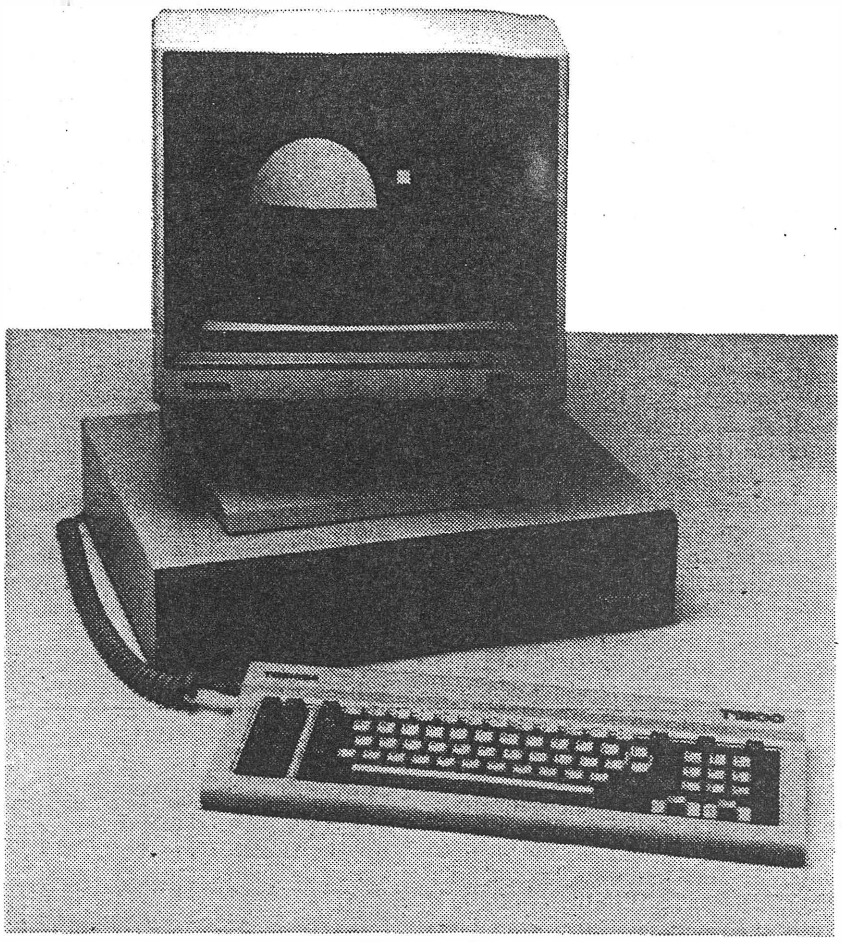
1 of 4.

expansion interface, which connects to their own range of printers, disk drives and other add-ons.

Most home computers use a standard domestic television for the display. A good colour television can give very respectable results, but for serious work you will soon be tempted to invest in a monitor. This is particularly true with serious word processing where you need to be able to display 80 columns of type on the screen, as domestic televisions are just not capable of this sort of resolution. A monitor requires a monitor output, and there are a variety of formats.

Memory is important. Programs, whether bought in cassette form or typed in yourself, take up memory, or RAM, as it is known in the trade. The more memory there is the longer and more interesting the programs the computer can handle. Advertisers make much of memory capacity, quoting '64k', '32k' from all directions. However this is not quite what it seems, as most computers use a substantial amount of RAM for the screen display and other internal purposes, leaving substantially less for your own use. High resolution, multi-coloured graphics are particularly demanding, so if you are concerned about memory make sure you are comparing 'user-available RAM', and not simply the quoted figures. High resolution graphics are all very well, but most televisions are unable to resolve much above 320 x 256 so anything higher is wasted without a monitor.

Finally, look at the Basic, the language that the computer uses, and the keyboard. The quality of the Basic is only relevant if you intend to write your own programs, but there are a horrific number of different 'dialects'.



Business Micros

The secret to buying a business micro is in the software. It is the software that tells it what you want it to do, so it is with the software that your decisions should start. The first step is to specify on paper as fully as possible the jobs you want the computer to do and the amount and type of data you want it to handle.

You will soon come across terms like 'CP/M compatible' or 'for MS-DOS' as you investigate software. CP/M and MS-DOS are two disk operating systems in common use over a wide range of computers. Unfortunately the compatability of CP/M or MS-DOS systems between machines depends on the type and capacity of the disk drives, so CP/M software for, say, a Kaypro is unlikely to run on an Epson. The IBM PC and the Apple II have become standards, and when a micro is described as, say, 'IBM compatible' it means that it should run all the software written for the IBM PC. But do check this before committing yourself!

Now we come to the hardware itself. There are several types of storage medium: the average floppy disk is capable of storing around 350 kilobytes, which is roughly equivalent to 60,000 English words, or about 60 pages of *PCG*. Some micros are capable of storing up to 1Mbyte (1,000k) per disk, but if your demands are more you will need to go for a hard disk or Winchester. These can store anything up to 20Mbyte, but at a substantial extra cost.

A big advertising point at present is whether the machine is 8-bit or 16-bit, 16-bit being supposedly faster and better. The difference in speed between the two is unlikely to be noticeable. The CP/M operating system was designed for 8-bit words, while MS-DOS is 16-bit, but most modern business micros tend to be 16-bit, though some are capable of running both.

The keyboard and screen are important. Make sure you are comfortable with both, particularly the keyboard. Some micros have a large number of 'function' keys, keys which do nothing in themselves, but can be programmed to carry out complex functions at a single keystroke. These are very useful for complex wordprocessors and databases.

Finally, there are alternatives to the usual three-box business micro. There are several so called 'portable' micros, with screen built in, though they are heavy enough to cause hernias! Handheld computers are becoming very popular as electronic notebooks, and can be linked back to your office computer from anywhere in the country, via modem. At the other end of the market are the multiuser systems, which allow several people with their own keyboard and screen to use the same database and share information. These are expensive if you only have two or three users, but if you are likely to expand they could well become cost effective.

Buying a micro is a costly and time consuming process. These tables can help you compare different models, but a good dealer is vital if you are to make the right decision.

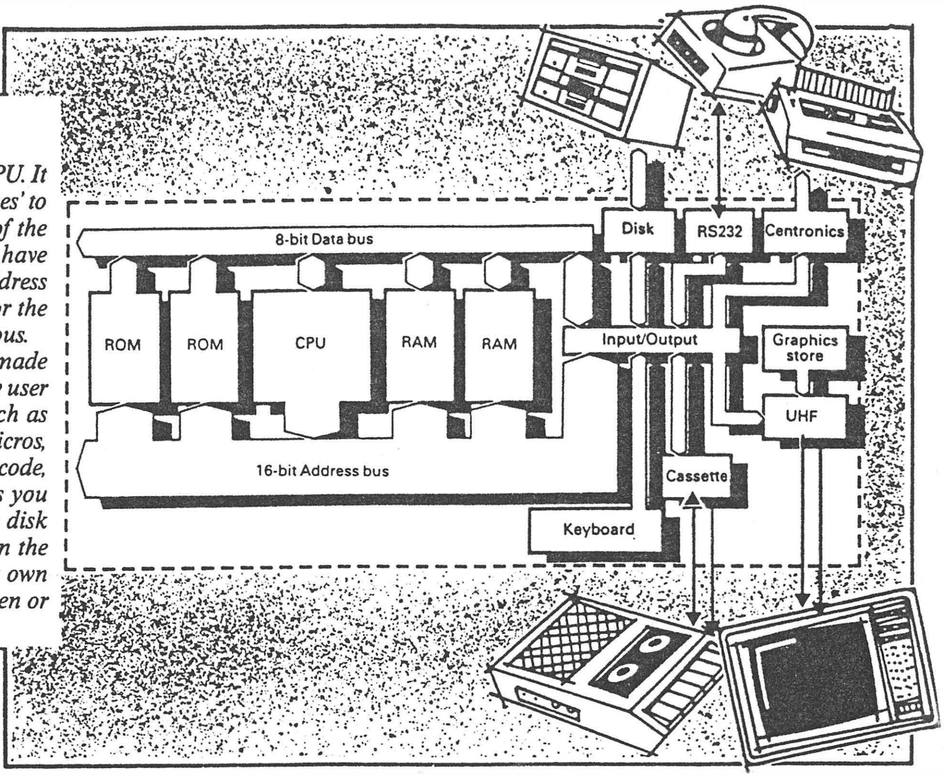
Dec/Jan 86 3(1)

2 of 4.

Inside your Micro

The heart of your micro computer is the microprocessor, or CPU. It is here that instructions are carried out. It is linked by two 'buses' to the computer's memory and the outside world. Every part of the memory and the components that deal with the outside world have a unique 'address'. By sending out an address along the address bus the CPU can call or send data to or from the memory or the input and output device. This data travels along the data bus.

The CPU itself talks in a language called 'machine code', made up of binary numbers. It is a very fast language but is not very user friendly, so we find it easier to use a high-level language such as Basic. This is by far the most common language on home micros, and the translating program, that converts Basic to machine code, is stored in the ROM so it cannot be altered. Any programs you enter, either directly by keyboard, via the cassette deck or disk drive, or even sent by telephone via the modem, are stored in the RAM. However the computer itself also uses the RAM for its own purposes, in particular to store the data required for the screen or TV display.



KEY TO HOME MICROS

Hardware

MAKE AND MODEL: The micros are listed in order of price.
PRICE: Includes tax.
CPU TYPE AND SPEED: Indicates the type of micro processor used and the clock rate.

Memory

STANDARD RAM: Amount of memory in k (1 kilobyte equals 1024 bytes). RAM that is dedicated to the screen display is included.
MEMORY EXPANSION: The amount of extra memory available and the price of the expansion.
MEMORY FOR BASIC: The amount of memory that is available to you when programming in Basic.

Graphics

Many home micros allow several 'modes' of graphics display to overcome their limited memory. A high resolution display usually goes hand in hand with a limited range of colours and little memory left over for Basic. A lower resolution allows more colours on the screen and more memory for your program. The tables lists the parameters for the highest resolution mode on top and the lowest resolution mode below.
GRAPHIC RESOLUTION: The number of points across and down that can be accessed individually.
COLOURS: The number that can be displayed at the same time, though they may be from a larger palette.
TEXT FORMAT: Number of characters that can be displayed across and down the screen. Eighty-character lines are needed for serious wordprocessing.
SPRITES: These useful programming tools enable you to move graphic designs easily around the screen. The maximum number available is shown.
USER-DEFINED GRAPHICS: Means you can redefine some of the keys on the keyboard to your own graphic motifs.

Keyboard

KEYBOARD TYPE: An F means a full typewriter keyboard, while C is a rubber pad calculator-type.
NUMERIC KEYPAD: This is a set of numeric keys grouped together.
FUNCTION KEYS: This is the number of keys that can be programmed to perform useful functions.

Basic

BASIC: An entry of BBC means BBC Basic, while MSX indicates the Basic conforms to the MSX standard. Otherwise a star rating out of three is given.
KEYWORDS: Indicates that a single keystroke enters a whole Basic command.
EDITOR: There are two types of Basic editor: is the more flexible and allows you to edit your programs anywhere on the screen L means you must select a line of your program into an editing area and alter it there.

Sound

NO. OF CHANNELS: This is the number of separate sounds that can be individually controlled.
ENVELOPE: Indicates that the amplitude of the sound can be fully controlled by an ADSR envelope.

Interfaces

CASSETTE: A cassette deck is usually required to store your programs. A ● indicates the player is built in, $ that you have to buy one (but any will do), and ρ indicates that the micro will use only its own-brand player.
DISK: Means that there is an interface to a disk drive built-in, $ means there is an interface but it costs extra.
RS232: The standard serial inteface used for communications, though some printers have optional RS232 interfaces. A $ means that it costs extra.
JOYSTICK: A ● indicates that the micro takes the near-standard Atari-type joystick, 0 indicates a different type of joystick.
MONITOR: Interfaces to monitors, which will give you a far clearer display than the home TV, come in two flavours: R indicates RGB and C composite.
EXPANSION: If a particular expansion interface is necessary.

Comments

BUSINESS EXPANSION: Many home micros can be expanded to a full business system.

Dec/Jan 86 3(1)

3 of 4

HOME MICROS

Hardware				Memory		Graphics					Keyboard			Basic			Sound		Interfaces								Comments		
Make and Model	Price	CPU type Speed	Standard RAM	Memory Expansion	Memory for Basic	Graphic resolution	Colours on screen	Text format Max/min	Sprites	User defined graphics	Keyboard type	Numeric keypad	Function keys	Basic	Keywords	Editor	No. of channels	Envelope	Cassette	Disk	Centronics	RS232	Joystick	Monitor	Expansion	Business Expansion	Software Available	Verdict	Supplier
VZ-300	$99	Z80 3.54MHz	18k	16k/ $84.50	18k	128x64 64x32	9	32x18			F			•	●	C	1		S	S	S	S	S	C	Disk drive		Limited range of software	Dated machine	Dick Smith Electronics (02) 888 3200
Commodore 16	$149	7501 1MHz	16k		2k 12k	320x200	12	40x25		●	F		8	••		C	2		●	●			S	C	Expansion port for Commodore add-ons		Limited	Comes with cassette deck and four games.	Commodore (02) 427 4888
Tandy TRS-80 Color Computer	$180	6809 1MHz	16k	32k/ 64k/	14k	256x192	8	32x16			F			•		C	1		S	●		●	●		Good range of own brand add-ons		Lots of educational software and good W/P	Dated machine. Extra RAM includes extended Basic	Tandy (02) 675 1222
Commodore Plus/4	$349	7610 1MHz	64k		50k 60k	320x200	16	40x25		●	F		8	••		C	2		0	●			●	C	Own serial interface Expansion port		Little to date	Built in business packages — not powerful.	Commodore (02) 427 4888
Sinclair ZX Spectrum Plus	$399	Z80 3.5MHz	48k		39k	256x176	16	32x24		●	F			••	●	C	1		S	$		S	S		Wide range of independent add-ons		Excellent range for education and games	Great way of getting into micros. QL style keyboard	TCA (02) 406 5533
Spectravideo SVI-728	$399	Z80A 3.58MHz	64k	64k/$299 up to 1Mb	28k	256x192 68x48	16	40x24	32	●	F	●	5	MSX	●	C	3	●	S	S	●	S	S	C	Disk, MSX cartridge ports, graphics tablet	●	Limited range of good games. CP/M programs available.	Full MSX micro	Rose Music (03) 899 2388
Commodore 64	$499	6510 1MHz	64k		38k	320x200	16	40x25	8	●	F		4	•		C	4	●	0	●			●	R	Own bus for ROM packs Own serial		Excellent range-games and business.	Poor Basic. Price includes cassette deck and four games.	Commodore (02) 427 4888
MicroBee	$499	Z80A 3.375MHz	32k		31k	512x242 128x48		80x24 64x16		●	F			••		L	1		S		$	●	S	R	Upgrade to Computer-in-a Book, or 128k model. Colour option ($135), and sound (3 channel) $120.		Mainly educational and games	Dated but can upgrade.	Applied Technology (043) 242 711
Sony HB75	$499	Z80 4MHz	80k		28k	256x192	16	40x24 32x24	32		F		10	MSX		L	3		S	●	●		●	C R	MSX cartridge port		Little to date — will be more soon	Full MSX specification micro	Sony (02) 887 6666
Toshiba HX10	$449	Z80 4MHz	80k		28k	256x192	16	40x24 32x24	32		F		10	MSX		L	3		S	●	●		●	C	MSX cartridge port		Little to date — will be lots more	Full MSX specification micro, probably the cheapest.	Toshiba (02) 887 3322
Amstrad CPC 464	$579	Z80 4MHz	64k		42k	640x200 160x200	2 16	80x25 20x25			F	●	12	•••		C	4	●	●	●	●	S	●	●	Built in cassette but no i/face for another		Good quality and more all the time	Comes with monitor and cassette. Colour monitor version $829	AWA Thorn (02) 638 8444
Commodore C128	$699	Z80 8052	128k		112k	320x200	16	80x25 40x25	8	●	F	●	8	•••		C	3	●	S	●			●	C	Fast disk drive with CP/M for $499		Runs CP/M mode and Commodore 64	Good buy, if a little expensive.	Commodore (02) 427 4888
Sinclair QL	$799	68008 7MHz	128k		92k	512x256	8	85x25			F		5	••		C	1			S	S	S	S	R	Own serial ports 2 Microdrives fitted		Supplied with 4 Psion business packages	Microdrives not proven yet. Not quite a serious business micro	TCA (02) 406 5533
Amstrad CPC 6128	$800	Z80 4MHz	128k		106k	640x200 160x200	2 16	80x25 20x25			F	●	12	•••		C	4	●	S	●	●	S	●	●	Built in 170k disk drive	●	CP/M system included — good range	Comes with monitor and disk drive. Colour monitor version $1000	AWA Thorn (02) 638 8444
MicroBee Computer-in-a-Book	$995	Z80A 3.375MHz	64k		31k	512x242 128x48		80x24 64x16		●	F			•• ••		L	1		S	●	S	●	S	R	Upgrade to 128k model. Colour option ($135), and sound (3 channel) $120	●	CP/M included with disk system	Expensive, but good.	Applied Technology (043) 242 711
Spectravideo X'Press	$999	Z80A 3.58MHz	64k	64k/$299 up to 1Mb	28k	256x192 68x48	16	80x24 40x24	32	●	F		5	MSX	●	C	3	●	S	●	●	●	S	C	Disk drive, MSX cartridge port	●	Bundled CP/M	Full MSX micro	Rose Music (03) 899 2388
Acorn BBC Model B+	$1302	6502 2MHz	64k		28k	160x256 640x256	8 2	20x32 80x32		●	F		10	BBC		C	3	●	S	●	●	●		R C	Tube, 4A-D channels, Bus. User port etc.	●	Compatible with old BBC	Expanded BBC, but still overpriced includes disk interface	Barson Computers (02) 888 9444
MicroBee 128k Small Business System	$1995	Z80A 3.375MHz	128k		31k	512x242 128x48		80x24 64x16		●	F			•••		L	1		S		S	●	S	R	Colour option $135, and sound option will give 3 channels for $120. Price include green or amber monitor.	●	All CP/M software system includes WordStar, Multi-plan, Mailmerge, Microsoft Basic and communication software.	Good machine for serious home use or small business.	Applied Technology (043) 242 711

Dec/Jan 86 3(1)

4 of 4.

Computers for the rest of us

What to buy until you win the lottery.

Peter Roberts reports on a computer designed for the rest of us.

A machine you can afford to buy and run - the VZ300.

ON TELEVISION they have the advertisement with the catchy jingle - "When I win the lottery". In my case that is going to be difficult because I never have enough money to buy lottery tickets. But if I did have all the money in the world no doubt I would buy a GeeWhizzBang Mark III and big note it to all my friends.

I'll be able to do that on the same day I become a space pilot and go exploring on Jupiter. Until that day comes I will have to make do with a computer designed for the rest of us. A machine I am able to afford, buy and run.

Second hand machines always seem to me, rightly or wrongly, to be a bit suss, so I opted for a new computer - possibly, probably the cheapest computer available in Australia, the VZ 300 from Dick Smith Electronics.

Basically, all I wanted to do was to learn to program and have a bit of fun as I did so without it costing me an arm a leg. For this the VZ300 fills the bill perfectly.

Viz the Wiz - why do we always give our computers names? Are we trying to make them human? Is the ultimate humanised computer a robot? Is this where it is all heading? - is made in Hong Kong - in Block 1 of the Tai Ping Industrial Centre in Ting Kok Road which is in Nam Hang which is in Tai Po which is in the New Territories which is in Hong Kong - and comes with an instruction manual full of the spelling mistakes you expect with a Hong Kong machine. There are pages all the way through which say, "This page is purposely left bank" - something to do with the students of Paris, I suppose.

But the thoughts it includes, the basic idea, the information it gives the reader are all sound. In the introduction there is a neat piece of computer philosophy which is worth repeating.

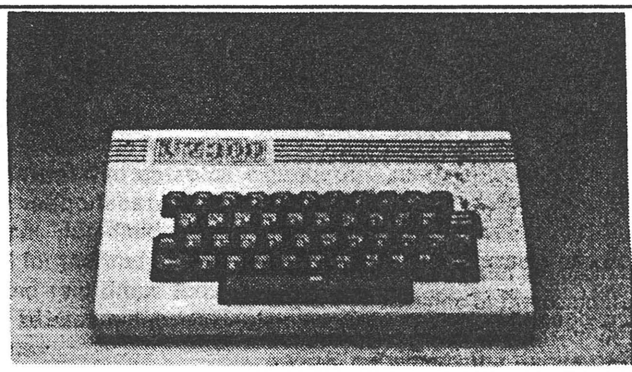
"The key to success is to try everything. It is not enough to read about it. You must do it. You don't learn to play the piano, type or swim by reading a book. You learn by doing. Don't worry about making mistakes. It is part of the learning process. If you make a mistake, just correct the mistake and continue. The computer doesn't worry about it, why should you? There is nothing that can be done from the keyboard that can damage your computer."

I have Viz lashed up to a colour television which has seen better days and every now and then gets the dreaded shudders with the screen giving a very neat imitation of a snowstorm at Chernobyl.

This doesn't worry me.

At least I have no one nagging at me that they want their daily ration of brain tranquiliser - "Prisoner", and will I please unplug my computer before I get done over.

So far my investment has only been for the computer, because the telly wasn't doing much anyway and I save and record the programs I



write on to a daggy old tape recorder which was given to my sister to encourage her to work harder at school but despite years of abuse from grotty "Australian Crawl" tapes nevertheless works perfectly OK.

I once read a manual for another computer from Tandy which suggested that it was wise to save everything three times if you were using a tape recorder. I know tape is not perfect but that is silly. I save twice on two different tapes so that when I am loading I don't have to work my way through programs plus their backups which would be a bit of a drag.

The great step plus of the Viz is that it uses one of the better forms of Basic - Microsoft. You know what you are using because when you buy the machine you have to sign all sorts of dire statements which say you will be, at least, severely damaged if you let anyone else use it.

Everyone in computing has a quiet sneer at Basic, which is to amateurs what Cobol is to professionals. And it is true it is not as snazzy as C or TurboLogic or even Pascal. (I am not too sure about the last one. All I have seen of Pascal makes me very suspicious).

But it is perfectly adequate and at least, providing you use lots of REMS, you can go back and work out why you wrote a particular sub-routine some months after you wrote it. Hands up anyone who can do that with the contradictory cryptic meanderings of C.

Because this is a colour computer I can produce quite pretty results without any particular problems and the handbook is extremely sensible in its instructions.

I am, of course, obsessed with winning on the horses and the fact that I can use decimals and some rather elegant mathematical formulas allows me to have hope that one day I will break the bookie's hearts.

Technically the Viz runs on a Z80A chip which was the pioneer of all those CP/M machines of yesterday. It is a brisk central processing unit and does not hang around especially when number crunching.

The Viz comes standard with 16K of memory although you can add expansion packs. This I refuse to do as I believe the only way to get tight well-written programs is to restrict the memory available. Shakespeare wrote his sonnets in the same way.

Dick Smith sell the Viz in quantities. They say they sell it mainly as a second machine to fathers who don't want their kids messing around with their serious computer.

I believe this underestimates the Viz and what it will do. It is a real computer at less than a hundred and fifty bucks. And it is a joy to program.

There are about seven zillion games programs available but as I like writing my own I haven't got round to testing them. Maybe next issue.

Below are listed a few of the programs you can get on cassette for the VZ300. You can see there is a good selection, and mostly the prices are really low:

Poker - $4.95. Straight draw poker - you can bet, raise, call, bluff and fold.
Hangman - $4.95. Educational game.
Blackjack - $4.95. Well known card game, played against the computer, with three other players if you like.
Circus - $9.50. You control two acrobats on a seesaw who must jump up and puncture floating balloons.
Biorhythm - $4.95. Calculates your emotional, physical and intellectual highs and lows.
VZ-Invaders - $9.50. Like the arcade favourite.
Matrix - $4.95. Allows manipulation of data stored in arrays.
Introduction to Basic - $13.95. Learn the commands and statements and how they are used.
Elementary Geometry - $4.95. Covers all the important equations to provide you with a geometrical calculator.
Speed Reading - $4.95. An easy-to-use method to improve your reading skills.
Planet Patrol - $13.95. the VZ version of a popular arcade game.

2 of 2.

1(1) 1986.

Dear Sir/Madam,

Congratulation on such a well written magazine. Never before have I been so taken with your descriptions of the games you review.

One such game review was 'Spitfire' and after reading it I suddenly became jealous of the fact that I could not use it on my VZ-300.

Your review on the VZ-300 in issue one was great. You even went so far as to say that there might be a VZ-300 games review in the next issue ... there wasn't!!

I would say that a lot of your readers who own VZ-200 and 300 computers feel the same. I would also say that a lot of your readers are VZ-200 and 300 owners.

When are you going to give us what we want? Don't you think we are part of the computer world as well?

So I call for all VZ owners to stand up and be counted!

David Kennedy
Campsie, NSW

ED: All those standing say aye!

1(4) Nov 86.

Tim Hartnell

While Halley's Comet was heading down towards Melbourne's West Gate Bridge, I was engrossed in writing a couple of books for Dick Smith's VZ-300. After a few hours of staring at fuzzy letters and numbers on my TV screen — which closely resembled the 'dissolving aspro' look of the aforementioned comet — I thought I'd try switching my reliable old Sharp 'Shot Vision' telly to its monitor setting (designed for those rich types who can afford VCRs). What a revelation. The old fuzzy letters, which suggested I'd been playing with my keyboard for so long that I was going blind, stood out in sharp clarity.

If the thought of indulging in a new monitor and/or graphics card is out of the question, Bankcard-wise, you could invest in an anti-glare screen.

Whenever I used the VZ nowadays, I send the picture to the 'monitor' setting of my TV. And the VZ picture is better than the image I get on my IBM Personal Computer Color Display, the official IBM monitor. Even as I write this on my PC I am aware that the letters I can see are somewhat indistinct approximations to the writing I should be observing.

All this is leading up to what I want to give my PC (and me) for Christmas. Top of the list is an upgraded graphics card so I can read text properly and a new monitor. (By the way, the eagle-eyed among you will see that in this article, I've stuck to products available from Dick Smith and/or Tandy stores, as these stores are well distributed throughout Australia, and Dick Smith goodies are available anywhere by mail. I thought there was no point in whetting your appetite with hard-to-acquire Chrissy gifts.)

In the monitor field, there are a lot to choose from, all which fit well within the $500 ceiling imposed by the editor on my

YC Dec 87 p. 20.
1 of 2.

Christmas wish list. Dick Smith's Hercules-compatible monographics adaptor (which also includes a second parallel printer port) is $299, and a suitable monitor (the Dick Smith 30 cm Green Screen job) is also $299. For Multitech and other computers with a fitted CGA, such as the VZ300, System 80/TRS80 or Apple II series and compatibles, Mr Smith's emporium also has a 30 cm monitor for $249, which is available in both ghastly green or sickly amber.

For $499, I could splash out on Tandy's Dual-Display Graphics Adapter which includes a Hercules-compatible setting to display up to 640 x 200 in 16 colours. Tandy also has a nice monitor, which is slightly more expensive than the Electronic Dick's. This is the VM-2 Monochrome Monitor, which comes with a 30cm green phosphor screen and a non-glare finish. Just outside the price range, if I'd wanted colour, is Tandy's CM-5 RGBI colour monitor, with a 33cm screen — ideal for 80 x 25 text and 320 x 200 graphics, for $599, while Mr Smith's 30 cm RGB colour monitor is $949.

The 'non-glare finish' mentioned above on the Tandy VM-2 is a real blessing. On the standard IBM monitor I've got, the screen has been polished so it resembles a mirror. Nothing is more distracting that trying to type while watching a reflection of yourself typing (especially when you look like I do). If the thought of indulging in a new monitor and/or graphics card is out of the question, Bankcard-wise, you could invest in an anti-glare screen.

I bought mine when I was in the UK, and it glories in the name of a 'Polaroid CP-50 contrast enhancement filter'. It cost, I seem to recall, around $70 a couple of years ago. Fortunately, if you want to reduce the reflections and glare from your own screen, and thus improve the visual output of your computer, in Australia Tandy have an anti-glare screen, which both reduces reflection and enhances contrast on a colour monitor, for $49.95. The improvement such a screen can make is extraordinary. Simply as a way of enhancing your pleasure at working with your computer, and in reducing the strain, few products can beat an anti-glare screen.

Now, once we've settled on the graphics card, monitor and (possibly) anti-glare filter, we need to sit our monitor on something. In the best traditions of do-it-yourself high tech, my IBM monitor is slanted up to the correct angle with a finely finished old one and a half inch by one foot lump of wood. You, of course, are entitled to more than this. Throwing the wood on the barbeque, you head for DSE and then decide if you want to be mean, not so mean, or foolishly profligate.

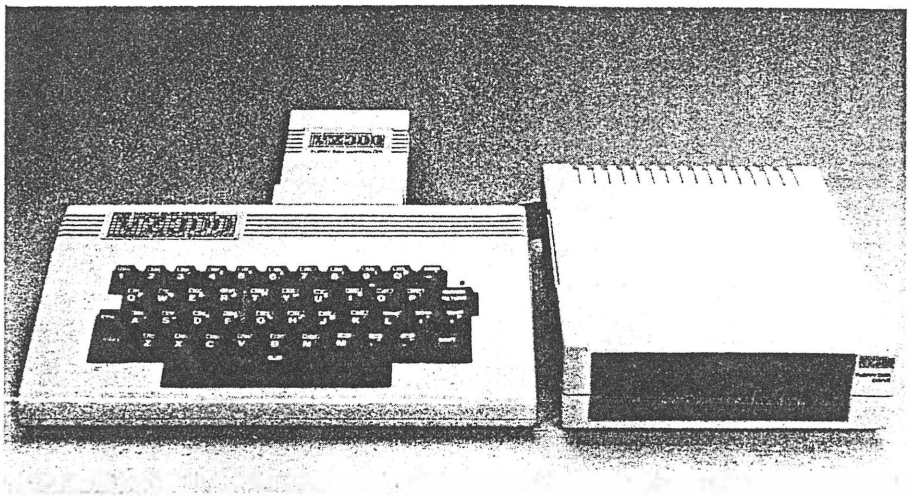
In the mean area, the deftly-named Budget Swivel Base is available for $24.95. This allows a full 360 degrees rotation, and 25 degrees vertical adjustment. A trifling $15 more and you could be the proud owner of the Deluxe Swivel Base, which has the added delight of a 'knurled knob on front for locking or freeing the monitor'. If you need to turn your monitor frequently, for example to allow others to see the screen, this is the one to choose. As I have often wished for a knurled knob, I think I would go for the Deluxe version.

After a smallish win on Tattslotto, I could invest in the Gas Lift Monitor Arm, an ergonomically-designed gas-lift arm which allows you to place the monitor exactly where you want it. To prove there is something up your sleeve, and to add to the neatness of your computer room, the monitor connection cables can be neatly concealed, inside the arm.

Midway between the mean and not-so-mean is Tandy's grandly-named, Universal Monitor Pedestal (which looks remarkably like Dick's non-knurled knob model), which features an adjustable rotating platform and the ability to tilt the monitor to the optimum viewing angle. A slightly more robust device, designed to fit under and then bend around the back so it comes over the back part of the keyboard unit of a computer, is Tandy's Monitor Platform for $99.95. This sturdy platform has a wooden top and metal legs.

The Hartnell menagerie of computers rests on a complex structure built from five wooden picnic tables from Myers. This is OK if you have a house like mine in which people and living plays second-fiddle to the demands of computers, but few people are willing to turn over all their living space to these silicon creatures. A Student's Computer Desk, to give your machine an established home, rather than the temporary resting place on the dining room table, is available from Tandy for $199.95, and measures 90 x 93 x 60 cm.

Those of the Tattslotto breed can splash out on a Deluxe Modular Workstation from Tandy, which consists of a system desk for $459.95, a storage hutch for a trifling $199.95, a Corner Section (now there's an imaginative name!) for $119.95 and a printer stand, with bottom-feed paper slot, fold-out paper catch and paper storage shelf, for $349.95. (Despite the



word 'deluxe' in the product name, there seems to be a noticeable shortage of knurled knobs with this combination, but I guess you can't have everything.)

The potential goodies go on and on. For printed output, Tandy has an 80 characters per second (cps) printer, the PC-compatible DMP-106, able to produce bit-mapped graphics, and using a 8 x 9 matrix, for $399.95. Tricky Dicky has a significantly faster printer; its 135 characters per second (draft mode), or 40 cps (near letter quality) dot matric print job for $499.00. This works on a 9 x 9 matrix for letters. For paper, Dick Smith seems to be cheaper, flogging off a box of 2,500 sheets of fanfold paper for $71.45, while Tandy's paper is $39.95 for 1000 sheets.

I think that's about it. I've now invested in a new graphics card and monitor, a deluxe support for my new monitor, some classy furniture to house my computer, a new printer and enough paper to write 'Tim Hartnell's War and Peace on your VZ-300'. All I need now is a knurled knob. □

YC Dec 87. 2 of 2.

DSE

THE LOW END of the Dick Smith Electronics (DSE) range of computers is the VZ-300 which epitomises the affordable computer — it's priced at less than $100. There is specific software developed for the VZ-300, but the range isn't large.

This is an excellent beginner's computer; the provision of Microsoft Basic allows the user to write programs with a minimum of learning. Also, it's predecessor, the VZ-200, still has thousands of enthusiastic users; (and a number of very active User Groups, which are an excellent source of information and public domain software on both computers).

In case you haven't already discovered it, the DSE catalogue is a worthwhile investment ($1). Not only does it cover its wide range of computers and add-ons, but many of the products listed are designed specifically for younger age groups.

Product Details

Product: VZ-300
Memory: 18 kilobytes RAM expandable to 34 Kbyte
Keyboard: 45 keys with auto repeat key.
Other: Data cassette; TV and Video output; Microsoft Basic builtin; options include joysticks, floppy disk drive, memory expansion kits and printer interface.
Price: $99 taxed

Figure 6. DSE's VZ-300 offers an almost painless introduction to computers.

YC Dec 87. p. 78.

GENERAL PROGRAMMING

Apr.	81	ETI	87-93	Extra Z80 opcodes.	(4)
Jun.	81	ETI	97	More uncovering Z80. (Dennis)	(1)
Jul.	81	ETI	83	Z80 uncovered. (Garland)	(-)
				Z80 CPU reference card	(2)
Feb.	82	YC	64-66	Understanding Assembler (Bell) Part I	(3)
Mar.	82	YC	74-77	(8080) Part II	(4)
Apr.	82	YC	61-63	" " " Part III	(3)
May	82	YC	60-62	" " " Part IV	(3)
Jun.	82	YC	99-101	" " " Part V	(3)
Jul.	82	YC	1-74	" " " Part VI	(3)
Sep.	82	YC	57-59	" " " Part VII	(3)
Nov.	82	YC	45-46	" " " Part VIII	(2)
Dec.	82	YC	93-97	" " " Part IX	(4)
Jan/Feb	83	YC	52-55	" " " Part X	(4)
Mar.	83	YC	61-62	" " " Part XI	(2)
Aug.	83	YC	62-68	" " " Part XII	(6)
Oct.	83	YC	87-89	" " " Part XIII	(2)
Nov.	83	YC	102-104	" " " Part XIV	(3)
Feb.	84	YC	93-94	" " " Part XV	(2)
Apr.	84	YC	123-126	" " " Part XVI	(2)
Nov.	82	PE	1/1-1/5	PE Micro-file #1 - 8080 & 8085 (Coles)	(5)
Jan.	83	PE	3/1-3/5	PE Micro-file #3 - Z80. (Coles)	(5)
Mar.	84	APC	73-85	Teach yourself assembler Pt. 1 (Overaa)	(6)
Apr.	84	APC	57-64	(8080, Z80, 6502) Pt. 2 (Overaa)	(5)
May	84	APC	89-98	" " Pt. 3 (Overaa)	(5)
Jun.	84	APC	53-60	" " Pt. 4 (Overaa)	(5)
Jul.	84	APC	61-64	" " Pt. 5 (Overaa)	(3)
Aug.	84	APC	110-116	" " Pt. 6 (Overaa)	(5)
Sep.	84	APC	145-151	" " Pt. 7 (Overaa)	(4)
Jan.	85	APC	122-124	Sort at input. (Ithell)	(1)
Feb.	85	APC	103-109	The basic art - algorithms, structures. (Liardet)	(4)
Mar.	85	APC	98-109	Pick a number - arithmetic. (Liardet)	(5)
Apr.	85	APC	79-87	It takes all sorts - sorting. (Liardet)	(5)
Oct.	85	APC	82	The Art of Programming - Progress. (Hjaltson)	(-)
Jun.	85	APC	170-171	Comment on binary search. (Lamich)	(1)
Jun.	85	APC	171-173	Comment on distribution sort. (Riordon)	(1)
Oct.	85	YC	107-8	Sorting out the sorts. (Jankowski)	(1)
Mar.	86	PE	17-18	Z80	(2)

Uncovering the Z80

Holmes and Watson would have been proud of the logic displayed in this investigation of one of computing's dark secrets.

THE Z80 is generally recognised as being just about the most powerful eight-bit micro around, and it's used in personal computers such as the TRS-80, the NASCOM and the Sharp MZ-80K. Zilog's literature for the Z80 describes its repertoire of 158 types of instruction, with a total of 696 possible opcodes (plus data).

You may think that this should be enough for anyone, but it's actually possible to find, on most Z80s, 88 more usable opcodes. These effectively give you access to four extra eight-bit registers; the more machine-code programming you do, the more you'll appreciate that you can't have too many registers.

This article explains what these instructions are and why they exist. It also gives a program which will test the Z80 in a TRS-80 to see if it possesses them.

The Z80 is a development of the Intel 8080A, from which it inherits the A-L registers. The second set of registers A^1-L^1) aren't in the 8080A, which also lacks IX and IY.

As well as the extra hardware, the Z80's designers also managed to cram in a lot more instructions. The Z80 can perform all the earlier micro's instructions, using the same opcodes, and has many more of its own. The extra instructions cover features such as bit testing, relative jumps, register shifts and block moves of data. Most importantly, as far as this article is concerned, they also provide a comprehensive set of indexed instructions.

These help to get round a curious limitation of the 8080A, inherited by the Z80, which is that a lot of references to memory have to use the register pair HL as a pointer. This sometimes leads to clumsy programming. For instance, to add the contents of address 1234H to the accumulator, we have to use:

```
LD    HL,1234H   ;HL=1234H
ADD  A,(HL)      ;A=A + DATA
```

The Z80 extends this type of addressing in order to have an indexing capability.

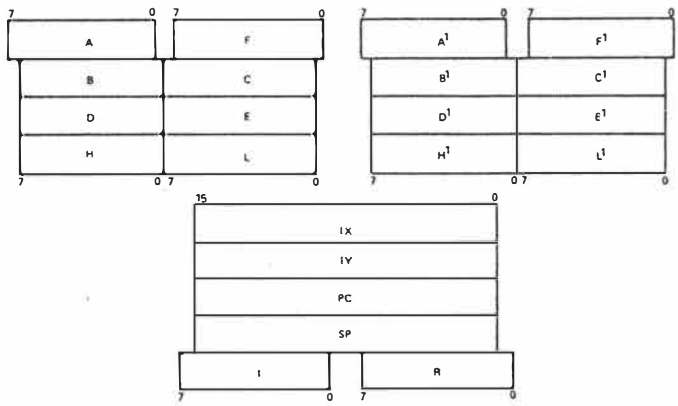


Figure 1. **What the Z80 looks like inside according to the manuals.**

Z80 architecture

To start, though, let's remind ourselves of the Z80's architecture. Figure 1 is a diagram of the micro.

The device has two sets of working registers, each set comprising a single accumulator (A), a flags register (F) and six general-purpose eight-bit registers (B-L); the six registers can be combined into three 16-bit registers. The micro has instructions to select the register set in use at any time.

The Z80 also has the usual program counter (PC) and stack pointer (SP), and two 16-bit index registers (IX and IY). We won't bother with I and R on Figure 1 here.

Indexed addressing

If you look at a description of the Z80's assembly-language, you'll soon see (I hope) something interesting about the way the micro does its indexing. Whenever an instruction has a form using (HL), it also has an indexed form. Thus we have:

```
LD    A,(HL)   LD    A,(IR+d)
BIT   7,(HL)   BIT   7,(IR + d)
```

I'm using 'IR' to represent 'IX or IY'. Furthermore, there are no indexed instructions which do not have (HL) counterparts.

I hope the suspicion is now growing that the two index registers and HL are closely related. This suspicion becomes a certainty when we look at the machine code which the micro actually executes.

For example, the Hex code to perform 'ADD A,(HL)' is 84; the equivalent code for 'ADD A,(IX + d)' is DD 84 dd, where 'dd' is the displacement expressed in two's complement form.

To take another example, the Hex code for 'BIT 7,(HL)' is CB 7E, and that for 'BIT 7,(IY + d)' is FD CB 7E dd. If you study your list of Z80 instructions (if you haven't got one, you shouldn't be reading this article!) you will see a remarkable consistency. Every (IX + d) instruction has an opcode formed by prefixing the equivalent (HL) command by 'DD', and adding 'dd' to the end. The (IY + d) commands are formed by using an 'FD' rather than 'DD' prefix.

This observation also partly explains why indexed instructions execute more slowly than their (HL) counterparts — the opcodes are two bytes longer. Reading the extra bytes takes time.

From this sort of evidence, I'm pretty certain that the Z80 uses the same internal logic to decode (HL) and (IR + d) instructions. The actual register selected is defined by the instruction's prefix, or lack of one.

Possibility of extra instructions

Having seen how the Z80 gets at its indexed instructions, an interesting possibility arises. So far, we've only considered HL as a 16-bit register, but it can, of course, be treated as two eight-bit registers. What happens if we take, say, the opcode for 'LD A,H' and prefix it with DD?

When I do it to the Z80 in my TRS-80, I find, amazingly enough, that A is loaded with the high byte of IX. No other registers have been altered. Lo and behold! I have an extra instruction. Obviously, it goes a lot further, or else I wouldn't be writing this!

On all the Z80s I've checked, the close relationship between HL, IX and IY allows each of the index registers to be treated for many purposes as two eight-bit registers. ▶

1 of 4

Since, in general terms, you can't have too many internal registers in a micro, this is potentially a very valuable discovery. Its usefulness obviously depends on whether or not you're using the index registers as index registers, but it gives an extra two eight-bit registers for each index register you can spare.

Extra instructions available

Let's have a look now at just what we can do with our extra registers. First of all, some nomenclature — I'll call the two bytes of IX 'XH' and 'XL', and the two bytes of IY 'YH' AND 'YL' (Figure 2). With these register names, we could, in the example above, use the mnemonic 'LD A,XH' for the instruction with the opcode DD 7C.

When I first discovered these extra commands, I hoped that XH etc. could be used in *any* Z80 operation that used H or L. For instance, we could have 'LD YL,B', 'SUB YH', 'CP XH', 'BIT 3,YL', etc. Unfortunately, the Z80 does not seem to work quite that way.

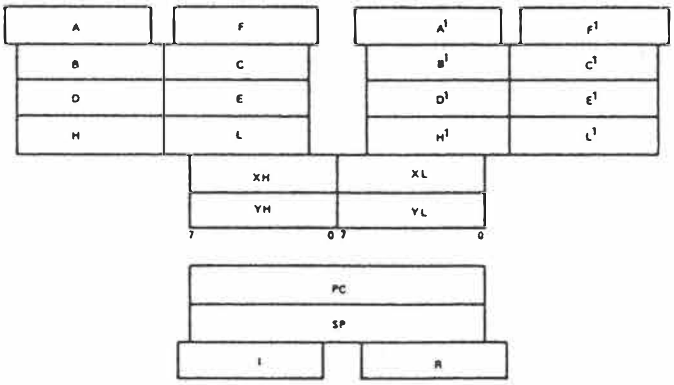


Figure 2. **What the Z80 might look like inside if you are lucky.**

In the first place, it's not possible to have, for example, 'LD XL,H'. This is not too surprising. The instruction would be generated by prefixing the code for 'LD L,H' (i.e: 6B) with DD. However, the micro would not know whether 'DD 6B' meant 'LD XL,H' or 'LD L,XH'; it actually settled on 'LD XL,XH'. So we cannot mix H or L with the extra registers in a single operation.

The second limitation is more obscure — i.e: I don't know why it exists! The extra registers will only work in the operations inherited from the 8080A, and not in the 'new' Z80-only instructions. As far as I can see, the difference is related to the fact that all the 8080A-compatible instructions use single-byte opcodes (plus data if it's appropriate), while the Z80 specials all use two bytes. Whatever the reason, it means that you can't use BIT, SET, RES, rotates or shifts. Still, the extra commands are free, so we can't complain.

Table 1 shows all the 'extra' instructions which are possible. It does not give their opcodes — you can form these by using the 'DD' and 'FD' prefixes as appropriate.

A small word of warning. I've shown the extra commands in the standard Z80 mnemonic format. However, it's no use trying them with your assembler, because it won't recognise them! You must either write a new assembler, or resort to hand coding.

It's important to remember that these extra instructions are 'unsupported'.

Mnemonic	Test Segment
LD r,XR	LD1
LD XR,r	LD2
LD XR,data	LD3
LD XR1,XR2	LD4
ADC A,XR	ADDSUB
ADD A,XR	ADDSUB
SBC A,XR	ADDSUB
SUB XR	ADDSUB
INC XR	INCDEC
DEC XR	INCDEC
AND XR	ANDORX
OR XR	ANDORX
XOR XR	ANDORX
CP XR	COMP

Notes:
'r' — Register A,B,C,D or E
'XR' — 'Register' XH,XL,YH or YL
'XR1', 'XR2' — Any XR
The mnemonics follow the usual Z80 conventions

Table 1. Extra instructions available.

That is to say, they don't appear in the official Z80 literature, and so there is no guarantee that every Z80 will execute them successfully. It may well be that, at some stage, Zilog will modify the micro's internal workings, and the change will stop it responding to these commands. Obviously, if a given chip obeys them once, it will obey them every time.

If you want to use them then you must test your micro to see how it responds to the opcodes. The best way is via a series of short machine-code program segments, preferably controlled via a high-level language such as BASIC so that you can evaluate the results easily.

Testing your micro

The first step in designing such a self-test program is to decide just what needs to be done. Is it, for example, necessary to check that 'LD A,XH', 'LD B,XH', 'LD C,XH', etc. *all* work properly? I think not. If we can show that, say, XH can be loaded into B, then it's virtually certain that it can be loaded into A, C, D and E also. It *is* worth checking that ▶

```
00100   ;ROUTINE TO CALL EACH TEST SEGMENT
00110   ;
00120   TSTALL  CALL  0A7FH          ;READ HL
00130           LD    A,75H          ;A = 75H
00140           LD    C,A
00150           LD    B,A            ;BC = 7575H
00160           LD    D,A
00170           LD    E,A            ;DE = 7575H
00180           CALL  7C45H          ;PERFORM TEST
00190           LD    (7C04H),BC     ;SAVE BC
00200           LD    (7C06H),DE     ;SAVE DE
00210           LD    (7C08H),IX     ;SAVE IX
00220           LD    (7C0AH),IY     ;SAVE IY
00230           LD    (7C02H),A      ;SAVE A
00240           JP    0A9AH          ;RETURN - PASS BACK HL
00250
```

Program 1. 'TSTALL'

```
00260   ;TEST THE 'LD R,XR' INSTRUCTIONS
00270   ;
00280   LD1     LD    IX,1234H       ;IX = 1234H
00290           LD    IY,5678H       ;IY = 5678H
00300           LD    B,XL
00310           LD    C,YH           ;BC SHOULD = 3456H
00320           LD    D,YL
00330           LD    E,XH           ;DE SHOULD = 7812H
00340           LD    A,XH         * ;A SHOULD = 34H
```


2 of 4

* see July 88 p 83

```
                RET

        TEST THE 'LD XR,R' INSTRUCTIONS

LD2     LD      BC,2345H        ;BC = 2345H
        LD      DE,7890H        ;DE = 7890H
        LD      XH,C
        LD      XL,D            ;IX SHOULD = 4578H
        LD      YH,A
        LD      YL,E            ;IY SHOULD = 7590H
        RET

        TEST THE 'LD XR,DATA' INSTRUCTIONS

LD3     LD      IX,0            ;IX = 0
        LD      IY,0            ;IY = 0
        LD      XH,17H
        LD      XL,23H          ;IX SHOULD = 1723H
        LD      YH,0F0H
        LD      YL,8BH          ;IY SHOULD = 0F08BH
        RET

        TEST THE 'LD XR1,XR2' INSTRUCTIONS

LD4     LD      IX,64H          ;IX = 0064H
        LD      XH,XL           ;IX SHOULD = 6464H
        LD      IY,3700H
        LD      YL,YH           ;IY SHOULD = 3737H
        RET

        TEST THE ARITHMETIC INSTRUCTIONS

ADDSUB  LD      A,90H           ;A = 90H
        LD      IX,8020H        ;IX = 8020H
        LD      IY,4030H        ;IY = 4030H
        ADD     A,XH            ;SHOULD BE: A = 10H, CY = 1
        ADC     A,XL            ;SHOULD BE: A = 31H, CY = 0
        SUB     YH              ;SHOULD BE: A = 0F1H, CY = 1
        SBC     A,YL            ;SHOULD BE: A = 0COH
        RET

        TEST THE 'INC & DEC' INSTRUCTIONS

INCDEC  LD      IX,OFFH         ;IX = OOFFH
        LD      IY,OFFOOH       ;IY = FFOOH
        INC     XH
        INC     XH
        DEC     XL              ;IX SHOULD = 02FEH
        DEC     YH
        DEC     YH
        INC     YL              ;IY SHOULD = FDO1H
        RET

        TEST THE 'LOGICAL' INSTRUCTIONS

ANDORX  LD      IX,OB51CH       ;IX = OB51CH
        LD      IY,96D4H        ;IY = 96D4H
        LD      A,O             ;A = O
        OR      XH              ;A SHOULD = B5H
        AND     YL              ;A SHOULD = 94H
        XOR     XL              ;A SHOULD = 88H
        RET

        TEST THE COMPARISONS

COMP    LD      IX,1234H        ;IX = 1234H
        LD      IY,5678H        ;IY = 5678H
        LD      A,34H           ;A = 34H
        CP      XH              ;A = XH?
        RET     Z               ;RETURN IF ERROR
        LD      A,56H           ;A = 56H
        CP      YH              ;A = YH?
        RET     Z               ;SHOULD RETURN FROM HERE
        LD      A,10H           ;SET ERROR CODE
        RET                     ;ONLY HERE ON ERROR
        END
```

Program 2. Test segments

```
10    REM TEST Z80 EXTRA INSTRUCTIONS
20    FL = -1: REM FL IS PASS/FAIL FLAG
30    CLS: PRINT @15, "TEST Z80 EXTRA INSTRUCTIONS":
40    POKE 16526,32:POKE 16527, 124:REM USR START POINT
50    FOR I = 31776 TO 31809:READ B:POKE I,B:NEXT:REM LOAD TSTALL
60    REM START TESTING
70    FOR I = 1 TO 8
80    READ IT,J1,J2,J3,J4,J5,F$:REM EXPECTED RESULTS AND CONTROL
      DATA
90    FOR I2 = 31813 TO 31812 + IT:READ B:POKE I2,B:NEXT:REM LOAD TEST
      SEGMENT
100   HL = USR (12345):REM RUN TEST
110   GOSUB 1000:REM RECOVER REGISTERS
120   IF A = J1 AND BC = J2 AND DE = J3 AND HL = 12345 AND IX = J4 AND
      IY = J5 THEN GOSUB 2000 ELSE GOSUB 3000
130   NEXT I
140   IF FL THEN PRINT@841, "TESTS OF EXTRA INSTRUCTIONS
      SUCCESSFUL"; ELSE PRINT@842, "TESTS OF EXTRA INSTUCTIONS
      FAILED";
150   END
1000  REM RECOVER REGISTERS
1010  REM A : 7C02H : 31746
1020  REM BC : 7C04H : 31748
1030  REM DE : 7C06H : 31750
1040  REM IX : 7C08H : 31752
1050  REM IY : 7C0AH : 31754
1060  A = PEEK(31746)
1070  BC = 256*PEEK(31749) + PEEK(31748)
1080  DE = 256*PEEK(31751) + PEEK(31750)
1090  IX = 256*PEEK(31753) + PEEK(31752)
1100  IY = 256*PEEK(31755) + PEEK(31754)
1110  RETURN
2000  REM SUCCESS MESSAGE
2010  PRINT@I*64,F$;:PRINT@I*64 + 8,"SATISFACTORY";
2020  RETURN
3000  REM SUBROUTINE TO PRINT ERROR INFORMATION
3010  PRINT@I*64 + 32,F$;:PRINT@I*64 + 40,"FAILED";:FL = O:REM SET
      BASIC MESSAGE AND FLAG
3020  PRINT@640,"FAILURE REPORT FOR SEGMENT";F $
3030  PRINT "REGISTERS:"TAB(19)"A" TAB(24)"BC" TAB(31)"DE" TAB(38)
      "HL" TAB(45)"IX" TAB(52)"IY"
3040  PRINT "SHOULD HAVE BEEN:" TAB(16)J1; TAB(22)J2, TAB(29)J3;
      TAB(36)12345; TAB(43)J4; TAB(50)J5
3050  PRINT "WERE:" TAB(17)A; TAB(22)BC; TAB(29)DE; TAB(36)HL,
      TAB(43)IX; TAB(50)IY
3060  PRINT@965, "PRESS 'A' TO ABANDON; PRESS 'C' TO CONTINUE";
3070  IN$ = INKEY$: IF IN$ = "" THEN 3070
3080  IF IN$ = "A" END
3090  IF IN$ = "C" PRINT@640,STRING$(191," ");: PRINT@832,STRING$
      (191," ");:RETURN
3100  GOTO 3070
4000  REM CALLING ROUTINE
4010  DATA 205, 127, 10, 62, 117, 79, 71, 87, 95, 205, 69, 124, 237, 67, 4, 124,
      237, 83
4020  DATA 6, 124, 221, 34, 8, 124, 253, 34, 10, 124, 50, 2, 124, 195, 154, 10
4030  REM LD1
4040  DATA 19, 52, 13398, 30738, 4660, 22136, LD1
4050  DATA 221, 33, 52, 18, 253, 33, 120, 86, 221, 69, 253, 76, 253, 85, 221, 125,
      201   * see July 81. p. 83
4060  REM LD2
4070  DATA 15, 117, 9029, 30864, 17784, 30096, LD2
4080  DATA 1, 69, 35, 17, 144, 120, 221, 97, 221, 106, 253, 103, 253, 107, 201
4090  REM LD3
4100  DATA 21, 117, 30069, 30069, 5923, 61579, LD3
4110  DATA 221, 33, 0, 0, 253, 33, 0, 0, 221, 38, 23, 221, 46, 35, 253, 38, 240, 253,
      46, 139, 201
4120  REM LD4
4130  DATA 13, 117, 30069, 30069, 25700, 14135, LD4
4140  DATA 221, 33, 100, 0, 221, 101, 253, 33, 0, 55, 253, 108, 201
4150  REM ADDSUB
4160  DATA 19, 192, 30069, 30069, 32800, 16432, ADDSUB
4170  DATA 62, 144, 221, 33, 32, 128, 253, 33, 48, 64, 221, 132, 221, 141, 253,
      148, 253, 157, 201
4180  REM INCDEC
4190  DATA 21, 117, 30069, 30069, 766, 64769, INCDEC
4200  DATA 221, 33, 255, 0, 253, 33, 0, 255, 221, 36, 221, 36, 221, 45, 253, 37,
      253, 37, 253, 44, 201
4210  REM ANDORX
4220  DATA 17, 136, 30069, 30069, 46364, 38612, ANDORX
4230  DATA 221, 33, 28, 181, 253, 33, 212, 150, 62, 0, 221, 180, 253, 165, 221,
      173, 201
4240  REM COMP
4250  DATA 21, 86, 30069, 30069, 4660, 22136, COMP
4260  DATA 221, 33, 52, 18, 253, 33, 120, 86, 62, 52, 221, 188, 200, 62, 86, 253,
      188, 200, 62, 16,201
```

Program 3. Program listing for the BASIC controller.

3 of 4

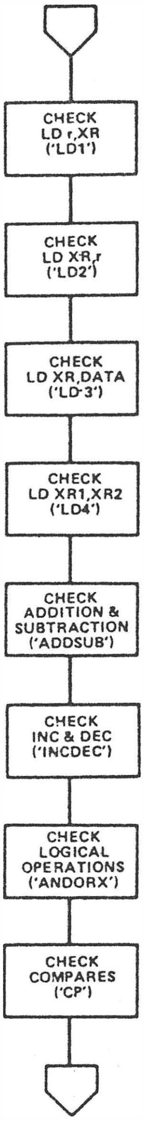


Figure 3. **Flowchart for the checking operations to find out if your Z80 has the 'added-extra'.**

each extra register can be loaded successfully into a normal register.

It is convenient for the program to check the extra instructions in logically-related blocks; I suggest that we can use the eight blocks shown in Table 1. Figure 3 shows the test sequence, which goes from the 'simpler' instructions to the 'more complex' ones.

Each block tests a suitable selection of the possible operations, and must do two things: it has to make sure that the extra operations work, and it has to check that the 'unused' registers are not corrupted. I decided that the best way to achieve these was to use a standard machine-code subroutine, which would call the test segments proper one at a time.

Before each test, all the registers in the micro would be set to known values and, at the end of the test, they would all be saved in memory. The high-level, controlling program (in BASIC) could then recover the stored data and test it for correctness before the next test.

Program 1 on page 88 is an assembly-language listing for this controlling subroutine ('TSTALL'), and Program 2 on pages 88-89 shows the eight test segments. All are written to suit a TRS-80 (Level II, 16K). Each segment is fairly simple, but a few comments are probably in order.

TSTALL. This segment starts with a 'CALL 0A7FH', and ends with 'JP 0A9AH'. These are the TRS-80 routines which pass the value of HL between BASIC and machine-code, via USR — by using these, I did not have to use TSTALL to store HL in memory.

This segment also uses a 'CALL7C45H' to get to each test segment; as we will see later, each is loaded, in turn, into the same area of RAM by the BASIC program. If the subsequent 'RET' goes wrong, then we know that SP has been corrupted by the tests.

ADDSUB. This segment tries each of the four eight-bit arithmetic operations once. I chose the values and the sequence of using them so that, as far as possible, multiple errors were unlikely to cancel each other out.

COMP. When we test the 'CP's, we have to make sure that the Z flag is set/reset at the right times. The 'LD's of A are arranged so that, if things go wrong, the segment exits with the wrong value in A.

Those, then, are the fundamental machine-code tests. To control them, however, I used a BASIC program, which made it much easier to assess the results and to format the output. The program has to do several things:

a. Load the appropriate machine-code segments.
b. Run the machine code.
c. Evaluate the results.
d. Output its assessment.

Program 3 on page 89 is a listing of the program that I used.

Initially, the calling routine is loaded into the top of memory by a series of READs and POKEs, and then the tests proper start.

The first line of DATA for each test segment defines the number of bytes in the subroutine, the expected values in all the registers except HL (which should always be 12345), and the title of the segment. This data allows the test segment to be loaded and run.

The actual values of the registers, saved in memory by 'TSTALL', are recovered by the subroutine at lines 1000-1100, and the result is evaluated. If the results are OK, a suitable message is printed, and the program goes on to the next test.

If any failure occurs, the subroutine at line 3000 is called. This prints out an error message, and the expected and actual data in the registers. The routine also clears a flag (FL) to show that there was a fault. Finally, the fault routine sits in a loop while you make up your mind what to do next.

Figure 4 shows the sort of display which might appear partway through the test of a Z80 which does not respond properly. You'll notice that I have to modify the 'expected' values to force a failure. At the end of the test, a success/failure message appears.

The only other point to watch out for when you run this program on a TRS-80 is the protection of the RAM used for the machine-code. There's probably no threat to it, but you should answer the 'MEMORY SIZE?' prompt with 31734 to be safe.

Use on other micros

The program here runs on a TRS-80. What, you may ask, do you have to do to run it on, say, an MZ-80K?

Obviously, the BASIC and the actual addresses used must be changed to suit the new machine. However, the critical parts of the program, the eight test segments, are all relocatable (they don't use absolute addresses), and so they shouldn't need any attention. You will have to massage 'TSTALL' a bit to suit how, or if, you pass the value of HL through a USR.

Conclusion

Most, if not all, Z80s have extra instructions in them which Zilog is very coy about. These instructions give the dedicated machine-code masochist four extra eight-bit general-purpose registers to play with, and can be very useful indeed.

It's very easy to test whether or not your micro has these commands. If it has, you've got an unexpected bonus, and if it hasn't — you never knew you were missing them. ●

```
TEST Z80 EXTRA INSTRUCTIONS

LD1     SATISFACTORY
LD2     SATISFACTORY
                      LD3      FAILED
LD4     SATISFACTORY
ADDSUB  SATISFACTORY
                      INCDEC FAILED

FAILURE REPORT FOR SEGMENT INCDEC
REGISTERS:  A     BC     DE     HL     IX    IY
SHOULD
HAVE BEEN:  117   32369  30069  12345  766   64769
WERE:       177   30069  30069  12345  766   64769

PRESS 'A' TO ABANDON: PRESS 'C' TO CONTINUE
```

Figure 4. **A typical failure output.**

4 of 4

Uncovering more of the Z80!

Holmes and Watson would have turned in their graves if they had read the article on uncovering the Z80 in April ETI, says reader Stephen Dennis of Dundas, NSW.

"It is evident that Dr Moriarty distracted the otherwise thorough investigation that was made, because several other undocumented instructions can be found.

"If one looks at the numerical order listing of Z80 op-codes in the back of the Zilog Assembly Language Programming Manual, a strange omission occurs between CB 2F and CB 38. After looking at the operations that occur, the following rotate instruction can be deduced (elementary, my dear Watson):

reg ♦ 2 x reg + 1

i.e: shift left once and add one, hence the new mnemonic:

RLO reg machine code: CB 30 to 37

where reg is any of A, B, C, D, E, H, L, (HL) (the machine code corresponds to the standard Intel/Zilog convention for register values, i.e: B=0 C=1 . . . (HL)=6 A=7).

"If one looks even harder at the other unused Z80 op-codes (those with ED as a prefix), it is possible for one to find even more op-codes. However, to date most of these are duplicates of other Z80 codes or have as yet unknown effects on the CPU (i.e: not so elementary, Watson).

"The best way to check these instructions is to try using them, because after all that is what the spirit of hobbyist computing is about (even if the manufacturers and advertisers tell you differently).

THE Z80 — NAKED AT LAST?

Following the article on 'Uncovering the Z80' in our April issue we published a note from Stephen Dennis in June issue's Printout (page 97) about further undocumented instructions. However, one reader, Mr Peter A. Schmektschek, uncovered what may be either an omission or two in the original article and/or differences between his System 80 and the original author's TRS80. Here's what Mr Schmektschek found:

"In the article 'Uncovering the Z80' on page 88 (of the April issue) I feel the source listing of program 1, line 340 should read LD A,XL and not LD A,XH.

"Also, on page 89, the source listing of program 2, line 4050 appears to have two bytes missing: line 4050 should read:
DATA 221,33,52,18,253,120,86,221,69,253,76,253,85,**221,92**, 221,125,201

"The instruction in bold (line 330 in program 1) was missing and so test segment 1 failed though the others were successful (on my System 80 Mk 1).

"Incidentally, in line 4040, program 2, the data 19, . . . will now be correct."

Another reader, Tony Garland of Terrey Hills, NSW, found he had to make the following changes for the program to work on his TRS80:

ADD NEW LINE
5 CLEAR 500
(to overcome ?OS error in 3090).
Alter line 4040 to read
4040 DATA 17,52,13398,30738,4660,22136,LD1
(otherwise, in the READ statement line 80, J4 will try to read LD2 in DATA line 4070, which it cannot do).

Tony found segment LD1 in his system failed.

1 of 1
(to accompany Apr 81 Article)

MICRO LOGIC CORP.

MLC

HACKENSACK, NJ

Z80 CPU

MICROPROCESSOR INSTANT REFERENCE CARD

Single-Byte-Opcode to Instruction Conversion

LSD ⇒

MSD	0	1	2	3	4	5	6	7
0	NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA
1	DJNZ n	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA
2	JR NZ,n	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA
3	JR NC,n	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST 00H
D	RET NC	POP DE	JP NC,nn	OUT (n), A	CALL NC,nn	PUSH DE	SUB n	RST 10H
E	RET PO	POP HL	JP PO,nn	EX (SP), HL	CALL PO,nn	PUSH HL	AND n	RST 20H
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR n	RST 30H

MSD	8	9	A	B	C	D	E	F
0	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	JR n	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR Z,n	ADD HL,HL	LD HL,(nn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR C,n	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET Z	RET	JP Z,nn	table	CALL Z,nn	CALL nn	ADC A,n	RST 08H
D	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	table	SBC A,n	RST 18H
E	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	table	XOR n	RST 28H
F	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	table	CP n	RST 38H

Multi-Byte-Opcode to Instruction Conversion

CB00	RLC B
CB01	RLC C
CB02	RLC D
CB03	RLC E
CB04	RLC H
CB05	RLC L
CB06	RLC (HL)
CB07	RLC A
CB08	RRC B
CB09	RRC C
CB0A	RRC D
CB0B	RRC E
CB0C	RRC H
CB0D	RRC L
CB0E	RRC (HL)
CB0F	RRC A
CB10	RL B
CB11	RL C
CB12	RL D
CB13	RL E
CB14	RL H
CB15	RL L
CB16	RL (HL)
CB17	RL A
CB18	RR B
CB19	RR C
CB1A	RR D
CB1B	RR E
CB1C	RR H
CB1D	RR L
CB1E	RR (HL)
CB1F	RR A
CB20	SLA B
CB21	SLA C
CB22	SLA D
CB23	SLA E
CB24	SLA H
CB25	SLA L
CB26	SLA (HL)
CB27	SLA A
CB28	SRA B
CB29	SRA C
CB2A	SRA D
CB2B	SRA E
CB2C	SRA H
CB2D	SRA L
CB2E	SRA (HL)
CB2F	SRA A
CB38	SRL B
CB39	SRL C
CB3A	SRL D
CB3B	SRL E
CB3C	SRL H
CB3D	SRL L
CB3E	SRL (HL)
CB3F	SRL A
CB40	see BIT
.....	see RES
CBFF	see SET

ED40	IN B,(C)
ED41	OUT (C),B
ED42	SBC HL,BC
ED43aa	LD (aa),BC
ED44	NEG
ED45	RETN
ED46	IM 0
ED47	LD I,A
ED48	IN C,(C)
ED49	OUT (C),C
ED4A	ADC HL,BC
ED4Baa	LD BC,(aa)
ED4D	RETI
ED4F	LD R,A
ED50	IN D,(C)
ED51	OUT (C),D
ED52	SBC HL,DE
ED53aa	LD (aa),DE
ED56	IM 1
ED57	LD A,I
ED58	IN E,(C)
ED59	OUT (C),E
ED5A	ADC HL,DE
ED5Baa	LD DE,(aa)
ED5E	IM 2
ED5F	LD A,R
ED60	IN H,(C)
ED61	OUT (C),H
ED62	SBC HL,HL
ED67	RRD
ED68	IN L,(C)
ED69	OUT (C),L
ED6A	ADC HL,HL
ED6F	RLD
ED72	SBC HL,SP
ED73aa	LD (aa),SP
ED78	IN A,(C)
ED79	OUT (C),A
ED7A	ADC HL,SP
ED7Baa	LD SP,(aa)
EDA0	LDI
EDA1	CPI
EDA2	INI
EDA3	OUTI
EDA8	LDD
EDA9	CPD
EDAA	IND
EDAB	OUTD
EDB0	LDIR
EDB1	CPIR
EDB2	INIR
EDB3	OTIR
EDB8	LDDR
EDB9	CPDR
EDBA	INDR
EDBB	OTDR

%%09	ADD XY,BC
%%19	ADD XY,DE
%%21aa	LD XY,aa
%%22aa	LD (aa),XY
%%23	INC XY
%%29	ADD XY,XY
%%2Aaa	LD XY,(aa)
%%2B	DEC XY
%%34d	INC (XY+d)
%%35d	DEC (XY+d)
%%36dn	LD (XY+d),n
%%39	ADD XY,SP
%%46d	LD B,(XY+d)
%%4Ed	LD C,(XY+d)
%%56d	LD D,(XY+d)
%%5Ed	LD E,(XY+d)
%%66d	LD H,(XY+d)
%%6Ed	LD L,(XY+d)
%%70d	LD (XY+d),B
%%71d	LD (XY+d),C
%%72d	LD (XY+d),D
%%73d	LD (XY+d),E
%%74d	LD (XY+d),H
%%75d	LD (XY+d),L
%%77d	LD (XY+d),A
%%7Ed	LD A,(XY+d)
%%86d	ADD A,(XY+d)
%%8Ed	ADC A,(XY+d)
%%96d	SUB (XY+d)
%%9Ed	SBC A,(XY+d)
%%A6d	AND (XY+d)
%%AEd	XOR (XY+d)
%%B6d	OR (XY+d)
%%BEd	CP (XY+d)

%% means DD or FD and for DD, XY means IX for FD, XY means IY

%%CBd06	RLC (XY+d)
%%CBd0E	RRC (XY+d)
%%CBd16	RL (XY+d)
%%CBd1E	RR (XY+d)
%%CBd26	SLA (XY+d)
%%CBd2E	SRA (XY+d)
%%CBd3E	SRL (XY+d)
%%CBd46	BIT 0,(XY+d)
%%CBd4E	BIT 1,(XY+d)
%%CBd56	BIT 2,(XY+d)
%%CBd5E	BIT 3,(XY+d)
%%CBd66	BIT 4,(XY+d)
%%CBd6E	BIT 5,(XY+d)
%%CBd76	BIT 6,(XY+d)
%%CBd7E	BIT 7,(XY+d)
%%CBd86	RES 0,(XY+d)
%%CBd8E	RES 1,(XY+d)
%%CBd96	RES 2,(XY+d)
%%CBd9E	RES 3,(XY+d)
%%CBdA6	RES 4,(XY+d)
%%CBdAE	RES 5,(XY+d)
%%CBdB6	RES 6,(XY+d)
%%CBdBE	RES 7,(XY+d)
%%CBdC6	SET 0,(XY+d)
%%CBdCE	SET 1,(XY+d)
%%CBdD6	SET 2,(XY+d)
%%CBdDE	SET 3,(XY+d)
%%CBdE6	SET 4,(XY+d)
%%CBdEE	SET 5,(XY+d)
%%CBdF6	SET 6,(XY+d)
%%CBdFE	SET 7,(XY+d)
%%E1	POP XY
%%E3	EX (SP),XY
%%E5	PUSH XY
%%E9	JP (XY)
%%F9	LD SP,XY

Hex and Decimal Conversion

LSD →

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	2
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	3
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	4
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	5
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	6
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	7
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	8
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	9
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	A
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	B
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	C
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	D
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	E
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Powers of Two

1	2	9	512
2	4	10	1,024
3	8	11	2,048
4	16	12	4,096
5	32	13	8,192
6	64	14	16,384
7	128	15	32,768
8	256	16	65,536

17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152
22	4,194,304
23	8,388,608
24	16,777,216

Unsigned Comparisons

example: CP B

A < B	JP C,YES
A ≤ B	JP C,YES ① JP Z,YES
A = B	JP Z,YES
A ≠ B	JP NZ,YES
A ≥ B	JP NC,YES
A > B	JR C,3 ① JP NZ,YES

YES represents label for code to be executed if condition is true. Internally, A-B is computed to determine flags as for 'SUB B'.

① Requires both instructions.

Status Flags

MSB — LSB

S	Z	-	H	-	P/V	N	C

S = Sign (MSB) of result
Z = 1 when result is Zero
H = Half carry from bit 3
P/V = 1 = Parity even for logic op or oVerflow for arithmetic op
N = 1 when last op was subtract (0 for add)
C = Carry (CY)

Interrupts and Reset

Falling edge sensitive $\overline{NMI}$ does a RST 66H regardless of IFF1, 2 (Interrupt Flip Flop).

If interrupts are enabled (IFF1=1), low level sensitive $\overline{INT}$ depends on mode:

MODE 0: Interrupting device puts instruction on bus (e.g. RST or CALL). Takes 2 extra time states.

MODE 1: Does a RST 38H (Z13).

MODE 2: Location pointed to by

15 — 8	7 — 1	0
I	M	0

and next hold vector of service subroutine. M (7 bit int vector index) is put on data bus by interrupting device (Z19)

IFF1 and IFF2 are both cleared by $\overline{INT}$ or DI. Both are set by EI. $\overline{NMI}$ clears IFF1. RETN loads IFF1 from IFF2. LD A,I and LD A,R set P/V flag to IFF2. Reset sets PC=0, IFF1=IFF2=0, I=0, R=0, MODE =0.

Pinout

A11	1	40	A10
A12	2	39	A9
A13	3	38	A8
A14	4	37	A7
A15	5	36	A6
Ø	6	35	A5
D4	7	34	A4
D3	8	33	A3
D5	9	32	A2
D6	10	31	A1
+5V	11	30	A0
D2	12	29	GND
D7	13	28	$\overline{RFSH}$
D0	14	27	$\overline{M1}$
D1	15	26	$\overline{RESET}$
$\overline{INT}$	16	25	$\overline{BUSRQ}$
$\overline{NMI}$	17	24	$\overline{WAIT}$
$\overline{HALT}$	18	23	$\overline{BUSAK}$
$\overline{MREQ}$	19	22	$\overline{WR}$
$\overline{IORQ}$	20	21	$\overline{RD}$

Registers

main		alternate		special	
A	F	A'	F'	I	R
B	C	B'	C'	INDEX IX	
D	E	D'	E'	INDEX IY	
H	L	H'	L'	STCK PTR SP	
small=8 Bit		large=16 bit		PGRM CTR PC	

A=Accumulator
F=Flags
I= Interrupt vector
R= Memory refresh

When AF,BC,DE,HL used as pairs A,B,D,H are high order.

ASCII Character Set

LSD		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	←	o	DEL

General Instruction Description (except shifts)

ADC x, y	Add y+CY to x.
ADD x, y	Add y to x.
AND x	AND x to A.
BIT b, x	Test bit b of x.
CALL c, x	If condition c is true call subroutine at x.
CALL x	Call subroutine at x (push PC and jump to x).
CCF	Complement carry flag.
CP x	Compare A with x (see "Unsigned Comparisons")
CPD	Compare A with (HL); DEC HL, DEC BC.
CPDR	Like CPD, but repeat until A=(HL) or BC=0.
CPI	Compare A with (HL); INC HL;DEC BC.
CPIR	Like CPI, but repeat until A=(HL) or BC=0.
CPL	Complement A (1's comp.).
DAA	Decimal adjust A (after add or sub of BCD data)
DEC x	Decrement x by 1.
DI	Disable interrupts.
DJNZ d	Decrement B; jump relative by d if B not zero.
EI	Enable interrupts after next instruction.
EX x,y	Exchange x with y.
EXX	Exchange BC, DE HL with BC', DE', HL'.
HALT	Halt (wait for interrupt or reset).
IM x	Set interrupt mode to x.
IN A, (n)	Input port n into A (6).
IN r, (C)	Input port (C) into r (7).
INC x	Increment x by 1.
IND	Load (HL) from port (C); DEC B; DEC HL; (7)
INDR	Like IND, but repeat until B=0 (7).
INI	Load (HL) from port (C); DEC B; INC HL; (7).
INIR	Like INI, but repeat until B=0 (7).
JP c, x	If condition c is true jump to location x.
JP x	Jump to location x.
JR c, d	If condition c is true jump relative by d.
JR d	Jump relative by d.
LD x, y	Load x with y (move y to x).
LDD	Load (DE) with (HL); DEC DE, DEC HL, DEC BC
LDDR	Like LDD, but repeat until BC=0
LDI	Load (DE) with (HL); INC DE, INC HL, DEC BC
LDIR	Like LDI, but repeat until BC=0
NEG	Negate A (2's comp.).
NOP	No operation.
OR x	OR x to A
OTDR	Like OUTD, but repeat until B=0 (7)
OTIR	Like OUTI, but repeat until B=0 (7)
OUT (C), r	Output r to port (C) (7)
OUT (n), A	Output A to port n (7)
OUTD	Output (HL) to port (C), DEC B, DEC HL, (7)
OUTI	Output (HL) to port (C), DEC B, INC HL, (7)
POP x	Pop x from top of stack updating SP
PUSH x	Push x onto top of stack updating SP
RES b, x	Reset bit b of x (to 0)
RET	Return from subroutine (pop PC)
RET c	If condition c is true return from subroutine
RETI	Return from interrupt
RETN	Return from NMI (see "Interrupts").
RST x	Call subroutine at x (1 byte inst)
SBC x, y	Subtract y+CY from x.
SCF	Set carry flag (to 1).
SET b, x	Set bit b of x (to 1)
SUB x	Subtract x from A
XOR x	XOR x to A

MICRO LOGIC CORP.
HACKENSACK, NJ

Z80 CPU

MICROPROCESSOR INSTANT REFERENCE CARD

MICRO CHART

Example of reading instruction set tables: ADC A,A ... ADC A,- entry says to see table; table shows opcode 8F, 4 states; and flag code 'A' which is defined under 'Flag Codes'. ADC HL,BC ... 2 byte opcode is ED,4A; flag code is H; takes 15 states. CALL C, address ... opcode is DC followed by 2 byte address; flag code is Z; states are described by note 5.

Instruction Set

ADC	A,—	TABLE	A
ADC	HL,BC	ED4A	H15
ADC	HL,DE	ED5A	H15
ADC	HL,HL	ED6A	H15
ADC	HL,SP	ED7A	H15
ADD	A,—	TABLE	A
ADD	HL,BC	09	G11
ADD	HL,DE	19	G11
ADD	HL,HL	29	G11
ADD	HL,SP	39	G11
ADD	IX,BC	DD09	G15
ADD	IX,DE	DD19	G15
ADD	IX,IX	DD29	G15
ADD	IX,SP	DD39	G15
ADD	IY,BC	FD09	G15
ADD	IY,DE	FD19	G15
ADD	IY,IY	FD29	G15
ADD	IY,SP	FD39	G15
AND	—	TABLE	C
BIT	—	TABLE	V
CALL	aa	CDaa	Z17
CALL	C,aa	DCaa	Z(5)
CALL	M,aa	FCaa	Z(5)
CALL	NC,aa	D4aa	Z(5)
CALL	NZ,aa	C4aa	Z(5)
CALL	P,aa	F4aa	Z(5)
CALL	PE,aa	ECaa	Z(5)
CALL	PO,aa	E4aa	Z(5)
CALL	Z,aa	CCaa	Z(5)
CCF		3F	G4
CP	—	TABLE	B
CPD		EDA9	T16
CPDR		EDB9	T(1)
CPI		EDA1	T16
CPIR		EDB1	T(1)
CPL		2F	N4
DAA		27	M4
DEC	(HL)	35	F11
DEC	(IX+d)	DD35d	F23
DEC	(IY+d)	FD35d	F23
DEC	A	3D	F4
DEC	B	05	F4
DEC	BC	0B	Z6
DEC	C	0D	F4
DEC	D	15	F4
DEC	DE	1B	Z6
DEC	E	1D	F4
DEC	H	25	F4
DEC	HL	2B	Z6
DEC	IX	DD2B	Z10
DEC	IY	FD2B	Z10
DEC	L	2D	F4
DEC	SP	3B	Z6
DI		F3	Z4
DJNZ	d	10d	Z(2)
EI		FB	Z4
EX	(SP),HL	E3	Z19
EX	(SP),IX	DDE3	Z23
EX	(SP),IY	FDE3	Z23
EX	AF,AF'	08	Z4
EX	DE,HL	EB	Z4
EXX		D9	Z4
HALT		76	Z4
IM	0	ED46	Z8
IM	1	ED56	Z8
IM	2	ED5E	Z8
IN	A,(C)	ED78	W12
IN	A,(n)	DBn	Z11
IN	B,(C)	ED40	W12
IN	C,(C)	ED48	W12
IN	D,(C)	ED50	W12
IN	E,(C)	ED58	W12
IN	H,(C)	ED60	W12
IN	L,(C)	ED68	W12
INC	(HL)	34	E11
INC	(IX+d)	DD34d	E23
INC	(IY+d)	FD34d	E23
INC	A	3C	E4
INC	B	04	E4
INC	BC	03	Z6
INC	C	0C	E4
INC	D	14	E4
INC	DE	13	Z6
INC	E	1C	E4
INC	H	24	E4
INC	HL	23	Z6
INC	IX	DD23	Z10
INC	IY	FD23	Z10
INC	L	2C	E4
INC	SP	33	Z6
IND		EDAA	P16
INDR		EDBA	Q(1)
INI		EDA2	P16
INIR		EDB2	Q(1)
JP	(HL)	E9	Z4
JP	(IX)	DDE9	Z8
JP	(IY)	FDE9	Z8
JP	aa	C3aa	Z10
JP	C,aa	DAaa	Z10
JP	M,aa	FAaa	Z10
JP	NC,aa	D2aa	Z10
JP	NZ,aa	C2aa	Z10
JP	P,aa	F2aa	Z10
JP	PE,aa	EAaa	Z10
JP	PO,aa	E2aa	Z10
JP	Z,aa	CAaa	Z10
JR	C,d	38d	Z(3)
JR	d	18d	Z12
JR	NC,d	30d	Z(3)
JR	NZ,d	20d	Z(3)
JR	Z,d	28d	Z(3)
LD	(BC),A	02	Z7
LD	(DE),A	12	Z7
LD	(HL),A	77	Z7
LD	(HL),B	70	Z7
LD	(HL),C	71	Z7
LD	(HL),D	72	Z7
LD	(HL),E	73	Z7
LD	(HL),H	74	Z7
LD	(HL),L	75	Z7
LD	(HL),n	36n	Z10
LD	(IX+d),A	DD77d	Z19
LD	(IX+d),B	DD70d	Z19
LD	(IX+d),C	DD71d	Z19
LD	(IX+d),D	DD72d	Z19
LD	(IX+d),E	DD73d	Z19
LD	(IX+d),H	DD74d	Z19
LD	(IX+d),L	DD75d	Z19
LD	(IX+d),n	DD36dn	Z19
LD	(IY+d),A	FD77d	Z19
LD	(IY+d),B	FD70d	Z19
LD	(IY+d),C	FD71d	Z19
LD	(IY+d),D	FD72d	Z19
LD	(IY+d),E	FD73d	Z19
LD	(IY+d),H	FD74d	Z19
LD	(IY+d),L	FD75d	Z19
LD	(IY+d),n	FD36dn	Z19
LD	(aa),A	32aa	Z13
LD	(aa),BC	ED43aa	Z20
LD	(aa),DE	ED53aa	Z20
LD	(aa),HL	22aa	Z16
LD	(aa),IX	DD22aa	Z20
LD	(aa),IY	FD22aa	Z20
LD	(aa) SP	ED73aa	Z20
LD	A,(BC)	0A	Z7
LD	A,(DE)	1A	Z7
LD	A,(aa)	3Aaa	Z13
LD	A,I	ED57	U9
LD	A,R	ED5F	U9
LD	A,—	TABLE	Z
LD	B,—	TABLE	Z
LD	BC,(aa)	ED4Baa	Z20
LD	BC,aa	01aa	Z10
LD	C,—	TABLE	Z
LD	D,—	TABLE	Z
LD	DE,(aa)	ED5Baa	Z20
LD	DE,aa	11aa	Z10
LD	E,—	TABLE	Z
LD	H,—	TABLE	Z
LD	HL,(aa)	2Aaa	Z16
LD	HL,aa	21aa	Z10
LD	I,A	ED47	Z9
LD	IX,(aa)	DD2Aaa	Z20
LD	IX,aa	DD21aa	Z14
LD	IY,(aa)	FD2Aaa	Z20
LD	IY,aa	FD21aa	Z14
LD	L,—	TABLE	Z
LD	R,A	ED4F	Z9
LD	SP,(aa)	ED7Baa	Z20
LD	SP,HL	F9	Z6
LD	SP,IX	DDF9	Z10
LD	SP,IY	FDF9	Z10
LD	SP,aa	31aa	Z10
LDD		EDA8	R16
LDDR		EDB8	S(1)
LDI		EDA0	R16
LDIR		EDB0	S(1)
NEG		ED44	B8
NOP		00	Z4
OR	—	TABLE	D
OTDR		EDBB	Q(1)
OTIR		EDB3	Q(1)
OUT	(C),A	ED79	Z12
OUT	(C),B	ED41	Z12
OUT	(C),C	ED49	Z12
OUT	(C),D	ED51	Z12
OUT	(C),E	ED59	Z12
OUT	(C),H	ED61	Z12
OUT	(C),L	ED69	Z12
OUT	(n),A	D3n	Z11
OUTD		EDAB	P16
OUTI		EDA3	P16
POP	AF	F1	Z10
POP	BC	C1	Z10
POP	DE	D1	Z10
POP	HL	E1	Z10
POP	IX	DDE1	Z14
POP	IY	FDE1	Z14
PUSH	AF	F5	Z11
PUSH	BC	C5	Z11
PUSH	DE	D5	Z11
PUSH	HL	E5	Z11
PUSH	IX	DDE5	Z15
PUSH	IY	FDE5	Z15
RES	—	TABLE	Z
RET		C9	Z10
RET	C	D8	Z(4)
RET	M	F8	Z(4)
RET	NC	D0	Z(4)
RET	NZ	C0	Z(4)
RET	P	F0	Z(4)
RET	PE	E8	Z(4)
RET	PO	E0	Z(4)
RET	Z	C8	Z(4)
RETI		ED4D	Z14
RETN		ED45	Z14
RL	—	TABLE	K
RLA		17	J4
RLC	—	TABLE	K
RLCA		07	J4
RLD		ED6F	L18
RR	—	TABLE	K
RRA		1F	J4
RRC	—	TABLE	K
RRCA		0F	J4
RRD		ED67	L18
RST	00H	C7	Z11
RST	08H	CF	Z11
RST	10H	D7	Z11
RST	18H	DF	Z11
RST	20H	E7	Z11
RST	28H	EF	Z11
RST	30H	F7	Z11
RST	38H	FF	Z11
SBC	A,—	TABLE	B
SBC	HL,BC	ED42	I15
SBC	HL,DE	ED52	I15
SBC	HL,HL	ED62	I15
SBC	HL,SP	ED72	I15
SCF		37	O4
SET	—	TABLE	Z
SLA	—	TABLE	K
SRA	—	TABLE	K
SRL	—	TABLE	K
SUB	—	TABLE	B
XOR	—	TABLE	D

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	
BIT 0,	CB,47	CB,40	CB,41	CB,42	CB,43	CB,44	CB,45	CB,46	DD,CB,d,46	FD,CB,d,46	V
BIT 1,	CB,4F	CB,48	CB,49	CB,4A	CB,4B	CB,4C	CB,4D	CB,4E	DD,CB,d,4E	FD,CB,d,4E	V
BIT 2,	CB,57	CB,50	CB,51	CB,52	CB,53	CB,54	CB,55	CB,56	DD,CB,d,56	FD,CB,d,56	V
BIT 3,	CB,5F	CB,58	CB,59	CB,5A	CB,5B	CB,5C	CB,5D	CB,5E	DD,CB,d,5E	FD,CB,d,5E	V
BIT 4,	CB,67	CB,60	CB,61	CB,62	CB,63	CB,64	CB,65	CB,66	DD,CB,d,66	FD,CB,d,66	V
BIT 5,	CB,6F	CB,68	CB,69	CB,6A	CB,6B	CB,6C	CB,6D	CB,6E	DD,CB,d,6E	FD,CB,d,6E	V
BIT 6,	CB,77	CB,70	CB,71	CB,72	CB,73	CB,74	CB,75	CB,76	DD,CB,d,76	FD,CB,d,76	V
BIT 7,	CB,7F	CB,78	CB,79	CB,7A	CB,7B	CB,7C	CB,7D	CB,7E	DD,CB,d,7E	FD,CB,d,7E	V
STATES:	8							12	20		

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	
RES 0,	CB,87	CB,80	CB,81	CB,82	CB,83	CB,84	CB,85	CB,86	DD,CB,d,86	FD,CB,d,86	Z
RES 1,	CB,8F	CB,88	CB,89	CB,8A	CB,8B	CB,8C	CB,8D	CB,8E	DD,CB,d,8E	FD,CB,d,8E	Z
RES 2,	CB,97	CB,90	CB,91	CB,92	CB,93	CB,94	CB,95	CB,96	DD,CB,d,96	FD,CB,d,96	Z
RES 3,	CB,9F	CB,98	CB,99	CB,9A	CB,9B	CB,9C	CB,9D	CB,9E	DD,CB,d,9E	FD,CB,d,9E	Z
RES 4,	CB,A7	CB,A0	CB,A1	CB,A2	CB,A3	CB,A4	CB,A5	CB,A6	DD,CB,d,A6	FD,CB,d,A6	Z
RES 5,	CB,AF	CB,A8	CB,A9	CB,AA	CB,AB	CB,AC	CB,AD	CB,AE	DD,CB,d,AE	FD,CB,d,AE	Z
RES 6,	CB,B7	CB,B0	CB,B1	CB,B2	CB,B3	CB,B4	CB,B5	CB,B6	DD,CB,d,B6	FD,CB,d,B6	Z
RES 7,	CB,BF	CB,B8	CB,B9	CB,BA	CB,BB	CB,BC	CB,BD	CB,BE	DD,CB,d,BE	FD,CB,d,BE	Z
SET 0,	CB,C7	CB,C0	CB,C1	CB,C2	CB,C3	CB,C4	CB,C5	CB,C6	DD,CB,d,C6	FD,CB,d,C6	Z
SET 1,	CB,CF	CB,C8	CB,C9	CB,CA	CB,CB	CB,CC	CB,CD	CB,CE	DD,CB,d,CE	FD,CB,d,CE	Z
SET 2,	CB,D7	CB,D0	CB,D1	CB,D2	CB,D3	CB,D4	CB,D5	CB,D6	DD,CB,d,D6	FD,CB,d,D6	Z
SET 3,	CB,DF	CB,D8	CB,D9	CB,DA	CB,DB	CB,DC	CB,DD	CB,DE	DD,CB,d,DE	FD,CB,d,DE	Z
SET 4,	CB,E7	CB,E0	CB,E1	CB,E2	CB,E3	CB,E4	CB,E5	CB,E6	DD,CB,d,E6	FD,CB,d,E6	Z
SET 5,	CB,EF	CB,E8	CB,E9	CB,EA	CB,EB	CB,EC	CB,ED	CB,EE	DD,CB,d,EE	FD,CB,d,EE	Z
SET 6,	CB,F7	CB,F0	CB,F1	CB,F2	CB,F3	CB,F4	CB,F5	CB,F6	DD,CB,d,F6	FD,CB,d,F6	Z
SET 7,	CB,FF	CB,F8	CB,F9	CB,FA	CB,FB	CB,FC	CB,FD	CB,FE	DD,CB,d,FE	FD,CB,d,FE	Z
STATES:	8							15	23		

	A(8)	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	
RLC	CB,07	CB,00	CB,01	CB,02	CB,03	CB,04	CB,05	CB,06	DD,CB,d,06	FD,CB,d,06	K
RRC	CB,0F	CB,08	CB,09	CB,0A	CB,0B	CB,0C	CB,0D	CB,0E	DD,CB,d,0E	FD,CB,d,0E	K
RL	CB,17	CB,10	CB,11	CB,12	CB,13	CB,14	CB,15	CB,16	DD,CB,d,16	FD,CB,d,16	K
RR	CB,1F	CB,18	CB,19	CB,1A	CB,1B	CB,1C	CB,1D	CB,1E	DD,CB,d,1E	FD,CB,d,1E	K
SLA	CB,27	CB,20	CB,21	CB,22	CB,23	CB,24	CB,25	CB,26	DD,CB,d,26	FD,CB,d,26	K
SRA	CB,2F	CB,28	CB,29	CB,2A	CB,2B	CB,2C	CB,2D	CB,2E	DD,CB,d,2E	FD,CB,d,2E	K
SRL	CB,3F	CB,38	CB,39	CB,3A	CB,3B	CB,3C	CB,3D	CB,3E	DD,CB,d,3E	FD,CB,d,3E	K
STATES:	8							15	23		

Flag Codes

	C	Z	P V	S	N	H	
A	C	Z	V	S	0	H	
B	C	Z	V	S	1	H	
C	0	Z	P	S	0	1	
D	0	Z	P	S	0	0	
E	=	Z	V	S	0	H	
F	=	Z	V	S	1	H	
G	C	=	=	=	0	U	
H	C	Z	V	S	0	U	
I	C	Z	V	S	1	U	
J	C	=	=	=	0	0	
K	C	Z	P	S	0	0	
L	=	Z	P	S	0	0	
M	C	Z	P	S	=	H	
N	=	=	=	=	1	1	
O	1	=	=	=	0	0	
P	U	F	U	U	1	U	(1)
Q	U	1	U	U	1	U	
R	=	U	F	U	0	0	(2)
S	=	U	0	U	0	0	
T	=	F	F	S	1	U	(3)
U	=	Z	F	S	0	0	(4)
V	=	Z	U	U	0	1	(5)
W	=	Z	P	S	0	U	
X							
Y							
Z	=	=	=	=	=	=	

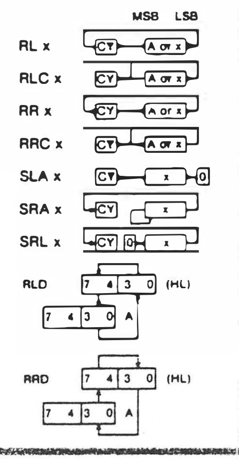
Codes:
0: reset
1: set
C: Carry*
F: Footnote
H: Half carry*
N: Add/Sub*
P: Parity*
S: Sign*
U: Undefined
V: oVerflow*
Z: Zero*
=: not affected

* Indicated flag affected by result

(1) Z=1 iff B becomes 0
(2) PV=0 iff BC becomes 0
(3) PV=0 iff BC becomes 0 and Z=1 iff A=(HL)
(4) PV=IFF2
(5) Z=$\overline{\text{bit}}$

	A	B	C	D	E	H	L	(HL)	n	(IX+d)	(IY+d)	
ADC A,	8F	88	89	8A	8B	8C	8D	8E	CE,n	DD,8E,d	FD,8E,d	A
ADD A,	87	80	81	82	83	84	85	86	C6,n	DD,86,d	FD,86,d	A
AND	A7	A0	A1	A2	A3	A4	A5	A6	E6,n	DD,A6,d	FD,A6,d	C
CP	BF	B8	B9	BA	BB	BC	BD	BE	FE,n	DD,BE,d	FD,BE,d	B
OR	B7	B0	B1	B2	B3	B4	B5	B6	F6,n	DD,B6,d	FD,B6,d	D
SBC A,	9F	98	99	9A	9B	9C	9D	9E	DE,n	DD,9E,d	FD,9E,d	B
SUB	97	90	91	92	93	94	95	96	D6,n	DD,96,d	FD,96,d	B
XOR	AF	A8	A9	AA	AB	AC	AD	AE	EE,n	DD,AE,d	FD,AE,d	D
LD A,	7F	78	79	7A	7B	7C	7D	7E	3E,n	DD,7E,d	FD,7E,d	Z
LD B,	47	40	41	42	43	44	45	46	06,n	DD,46,d	FD,46,d	Z
LD C,	4F	48	49	4A	4B	4C	4D	4E	0E,n	DD,4E,d	FD,4E,d	Z
LD D,	57	50	51	52	53	54	55	56	16,n	DD,56,d	FD,56,d	Z
LD E,	5F	58	59	5A	5B	5C	5D	5E	1E,n	DD,5E,d	FD,5E,d	Z
LD H,	67	60	61	62	63	64	65	66	26,n	DD,66,d	FD,66,d	Z
LD L,	6F	68	69	6A	6B	6C	6D	6E	2E,n	DD,6E,d	FD,6E,d	Z
STATES:	4							7		19		

Rotates and Shifts



Addressing

n	n is immediate 8-bit data.
aa	aa is immediate 16-bit data or address to CALL/to JP to.
(aa)	aa is address of data.
(rr)	16-bit reg rr holds address of data or address to CALL or to JP to.
(n)	n is port number.
(r)	8-bit reg r holds port number.
(IX+d)	IX+d is address of data (d is a 1 byte signed displacement).
d	In relative jumping, address to jump to is d + address of next instruction (d is signed).

Full 2 byte addresses in code, stack, and data areas are stored low byte followed by high byte. Thus JP 1234H is: C3,34,12.

SP points to used byte at top of stack. PUSH decrements SP by 2.

Intentionally Blank

Notes

(1) 21 except 16 at termination
(2) 13 except 8 at termination
(3) 12 for success; 7 for failure
(4) 11 for success; 5 for failure
(5) 17 for success; 10 for failure
(6) A to A15..A8 and n to A7..A0
(7) B to A15..A8 and C to A7..A0
(8) See faster version of 'Rotate A' instructions

2 of 2

UNDERSTANDING ASSEMBLER PART I

If you've mastered BASIC and feel you're ready for something with a bit more power, why not tackle assembly language? It's probably easier than you think, and LES BELL says it's capable of doing an awful lot.

your computer tutorial

BASIC is fine for a lot of jobs, but there are a lot of things it simply can't do — like high-speed bit fiddling, or input-output. That's why disk operating systems, BASIC interpreters and similar complex pieces of software are written in assembly language, not BASIC. And if you ever want to fix bugs in your operating system, or patch the input/output drivers, a knowledge of assembly language is indispensable.

What is assembly language? To understand that, it is necessary to start one level down, with an understanding of the microprocessor chip itself. In this series of articles, we'll learn to program the Z-80 microprocessor, the chip used in the TRS-80, System 80, ZX-81 and other popular microcomputers.

In fact, the Z-80 chip was a descendant from an earlier device, the Intel 8080. The Z-80 took the basic design of the 8080 and added extra registers and instructions, so it is still capable of running programs written for the 8080.

There are still a lot of 8080s around (I'm using one right now), and for this reason, we are going to deal with the 8080 subset of the Z-80. Besides, many readers will be using the CP/M operating system, which is supplied with an 8080 assembler and debugger as standard and which does not take advantage of the added facilities of the Z-80.

Scalpel, Please

Since we're going to be dealing with the actual bits and bytes of the computer, we'd better understand what a computer is, rather than what it does. Figure one shows the organisation of a typical microcomputer.

All the elements of the computer are linked together by a set of signal lines called a **bus**. This carries power to the various parts so they can work, and it carries a clock signal, which is the computer's 'hearbeat'.

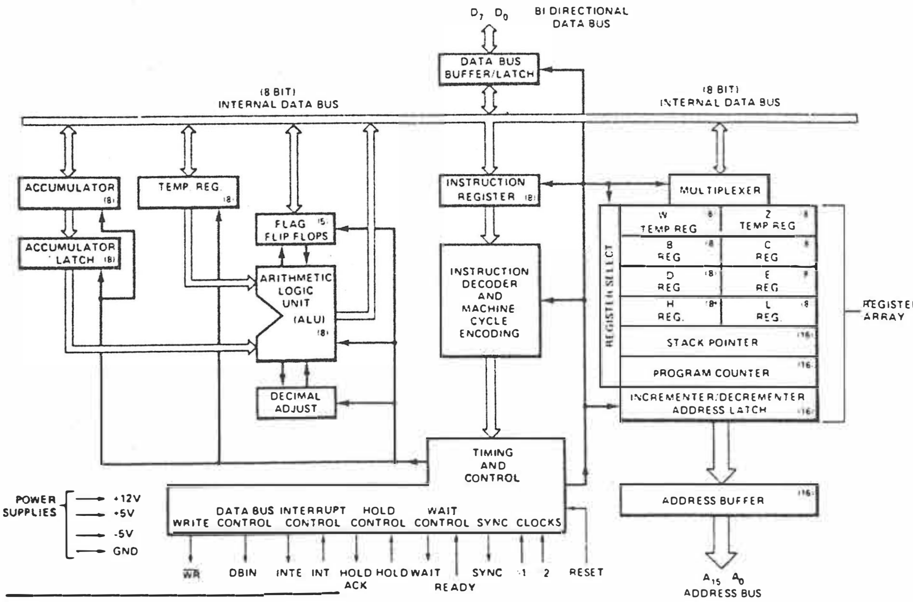
What is a microprocessor?

Leaving the electrical characteristics of the chip aside (we'll hardly refer to the electronic aspects at all), the chip is basically a set of registers which can store binary numbers, an arithmetic/logic unit which can process those numbers, and an instruction decoder which can analyse the program and get the various parts of the chip working together.

Figure one shows the various parts of the chip. At the left, you'll see the accumulator. This is a special register which works closely with the arithmetic/logic unit (ALU). You'll also see an accumulator latch and temporary register. Ignore these, as they're only used by the processor internally, and you can't get at them to do anything.

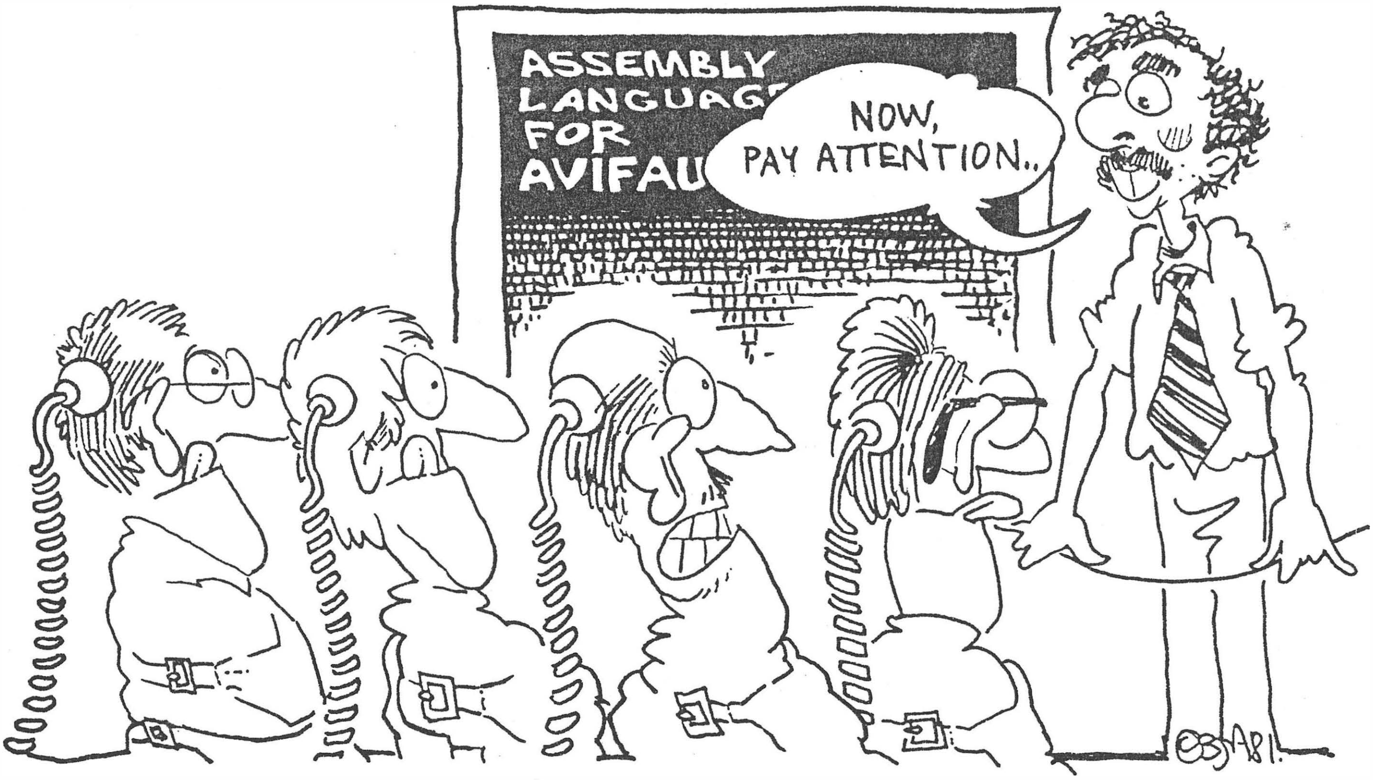
Attached to the ALU are five flag flip-flops. A flip-flop is a single memory cell which flips (or flops) from 0 to 1 and back again, depending upon certain conditions in the ALU logic to which it is attached. The flags tell us certain facts about the result of the last calculation performed by the ALU: whether the result is zero, negative or whether the calculation generated a carry and other useful info. Also attached to the ALU is a block of circuitry marked decimal adjust. We'll cover that when we do arithmetic.

On the right you'll see the register array, containing a number of 8-bit and 16-bit registers. The W and L registers are used internally, and we can't get at them. Registers B, C, D, E, H and L are 8-bit general purpose registers which can be operated on by a number of instructions. They can also be paired up to make the BC, DE and HL 16-bit register pairs, and



8080 CPU Functional Block Diagram

1 of 3. YC Feb 82 1(8) p64-66.



there are special 16-bit instructions which operate on them.

The stack pointer is a special purpose 16-bit register which is used in many different ways — we'll see most of these uses later. For now, it's enough to say the stack pointer is used by the processor to save temporary values.

Finally, the program counter is the 16-bit register which is used to fetch instructions from memory, sequentially. The remaining circuitry shown on the diagram can largely be relied upon to function automatically, without our having to worry about it.

Pulling it Together

How do all these registers and circuitry function together, and what do they do?

Let's examine how some of the simpler instructions work. We'll start by writing a simple program to add together two numbers which are stored in memory. This is done in the following way.

First, we load the first number into the accumulator (that's where all arithmetic is done) then add the second number to it. Finally we'll store the result back in memory. Here's the program:

```
LDA  NUM1
MOV  B,A
LDA  NUM2
ADD  B
STA  ANS
```

Step by step, here's how this program works: At the beginning of the program, the program counter points to the first instruction (LDA — Load Accumulator). It fetches it into the instruction decoding circuitry, which recognises it and organises the internal circuitry of the CPU to carry out the instruction.

This involves fetching the next two bytes following the instruction. Together, these bytes form the address where the first number is to be found. That's right — NUM1 is *not* the first number, but the address where the first number is to be found.

Having fetched this address from the program, the processor then puts the address out again, this time on the address bus, activating that memory location so the processor can read its contents and move it into the accumulator.

That completes the first instruction.

Contents Retrieved

We now have retrieved the contents of NUM1 and placed them in the accumulator. The program counter is pointing at the next instruction, and we are ready to execute it. The procedure is exactly the same — we fetch the instruction, identify it, and then execute it. In this case the instruction is a MOVe, into register B, from register A.

The purpose is to temporarily save our first number while we load the second number into the accumulator. This is necessary because the 8080 does not allow us to add a number directly from memory into the accumulator (with one exception, but later, later..). Nor does it allow us to move directly from memory into any register other than the accumulator (with one..).

So we have to load the second number into the accumulator, which means saving the first. Notice that the MOV instruction specifies the destination of the data first, then the source: MOV B,A means *into* B *from* A.

We can now use a LDA instruction again, this time to load the accumulator with the contents of location NUM2. We are now ready to perform the addition, which uses the ADD instruction to add the contents of B into the accumulator A.

We now have our answer, but it is in the accumulator. To store it back in memory, we use the STA (store accumulator) instruction to put it into the address specified. The processor executes this in just the same way as the LDA instruction — except when it puts out the address it writes into it instead of reading from it.

Language Characteristic

That's it. It seems like an awful lot of work, but that's characteristic of assembly language. Let's see how the program would be written for actual assembly and execution on a computer. Here's the full program:

YC Feb 82. 1(8) p 64-66 2 of 3.

```
;Program to add two bytes together
; 24/11/81

          ORG     0100H

START:    LDA     NUM1    ;Get first val
          MOV     B,A     ;Save it in B
          LDA     NUM2    ;Get second
          ADD     B       ;Add together
          STA     ANS     ;Store result

;Data storage area
NUM1      DB      27
NUM2      DB      13
ANS       DS      1

          END
```

The lines beginning with semi-colons (;) are comment lines, the same as REMs in BASIC. They have no effect on the generated code.

Assembly language lines are split into sections called **fields**. The first field of each line is called the **label** field. If we want to refer to a particular address, we label it by putting a name into this field. Later (or earlier) in the program we can refer to such locations symbolically, with such statements as JMP START, or LDA NUM1.

The second field contains the mnemonic for the instruction, and the third field contains the data or address it operates on; often called the **operand**. The fourth field contains an optional comment.

As well as the instructions we wrote out previously, there are a few others in the program. The mnemonics for the instructions are sometimes called **op-codes**; they are the instructions the computer will follow.

The new op-codes aren't really op-codes at all, which is why they're called **pseudo-ops**. Instead they're there to give the assembler program information it needs to assemble the program correctly.

For example, the ORG statement allows us to tell the assembler where in memory we would like this program located. It's short for ORiGin, and in this case says the program should start at location 0100H (the H stands for hexadecimal). I've chosen this address for compatibility with CP/M.

The Definitive Byte

Further down there are two DB statements. The DB (Define Byte) statement sets aside a single byte of storage, and initialises it to the value given in the DB statement. In this case, we've said NUM1 is a single byte of storage immediately following the program, and that it is to contain the value 27. NUM2 will immediately follow that, and will contain the value 13.

The next line contains a DS (Define Storage) pseudo-op. This is like the DB statement in that it sets aside memory for data storage, but it says nothing about what initial values these locations should have. Consequently, when the program starts running memory which has been reserved using a DS statement, it could contain anything. The DS 1 statement reserves one byte of storage; DS 32 would reserve 32 bytes, and so on.

I hope I don't have to explain what the END pseudo-op tells the assembler!

Now let's look at what the assembler outputs as a result of the assembly. There are two major files. One is called the **assembly listing** and contains our original input, with the machine code added into it. The other is called a **hex file** and contains the machine code alone, in a form suitable for machine loading. If our original file was called ADD.ASM, then running the assembler by typing ASM ADD will produce these two files, called ADD.PRN and ADD.HEX.

Here's ADD.PRN:

```
                    ;PROGRAM TO ADD TWO BYTES TOGETHER
                    ; 24/11/81

0100                          ORG     0100H

0100 3A0B01         START:    LDA     NUM1     ;GET FIRST VALUE
0103 47                       MOV     B,A      ;SAVE IT IN B
0104 3A0C01                   LDA     NUM2     ;GET SECOND
0107 80                       ADD     B        ;ADD TOGETHER
0108 320D01                   STA     ANS      ;STORE RESULT

                    ;DATA STORAGE AREA
010B 1B             NUM1      DB      27
010C 0D             NUM2      DB      13
010D                ANS       DS      1

010E                          END
```

As you can see, it looks just like the original, with the addition of two more fields on the left (I don't know why on the left, right seems more logical to me too!). The leftmost field contains the address of the first byte of the instruction or data. The next field contains the generated code, in hexadecimal.

Looking at the PRN file, we can see NUM1 is located at 010B hex. If you look at the first instruction, you'll see it reads:

```
0100 3A0B01    START:   LDA     NUM1
;GET FIRST VALUE
```

The 0100 is the address of the first byte of the instruction, which is a 3AH. Then follows the generated code: 3A0B01. We found NUM1 at 010BH; why are the next two bytes 0B01? The answer is that the 8080, for reasons known only to itself, reverses the order of the bytes in a 16-bit word, putting the least significant byte first. So you'll soon get into the habit of mentally swapping the bytes in this part of the listing.

Incidentally, there's a pseudo-op for words, analogous to DB for bytes, which swaps the two halves of the word. So

```
DB 01FFH
```

would generate code of

```
XXXX  FF01        DB 01FFH
```

automatically reversing the bytes.

Now here's the HEX file:

```
:0D0100003A0B01473A0C0180320D011B0D36
:0000000000
```

Notice it's not actually machine code, merely a hex dump of the program, with some added information to assist in loading the program. The exact details of the hex file don't matter at this stage; there's a CP/M program called LOAD which turns this file into an executable program (that is, a .COM file).

Next month we'll delve into the complete 8080 instruction set, then we'll be ready to write complete and sensible programs. By the way, there's a deliberate error in the program above: can you see what it is? □

 YC Feb 82 3 of 3.

UNDERSTANDING ASSEMBLER
PART II

your computer
tutorial

In part II of his Assembly Language series LES BELL introduces the instruction set of the 8080 microprocessor.

THE INSTRUCTION set of a computer falls into functional groups — data transfer instructions, arithmetic instructions and so on. This month we'll break down all the 8080 instructions into their functional groups, with a brief description of each for reference and a table giving their hex and decimal opcodes.

This installment in our series is not intended to impart a complete understanding of the 8080 assembly language. We simply want you to read through the instruction set and see the kinds of operations that are possible. Later, we'll start using them.

The instructions which move data from memory into and out of the processor are called the **data transfer group.**

All registers are eight-bit and register pairs can contain 16-bit values, often addresses, especially in the case of the HL pair.

Here are some abbreviations you'll need to know: **rdest** is the destination register (where the data is going to); **rsource** is the source register (...coming from); **bdata** is byte data (8 bits); **wdata** is word data (16 bits); **rp** is register pair.

There are three register pairs, BC, DE and HL. In assembly language, they are referred to by the first of the two registers, B, D and H.

MOV rdest,rsource: Moves the contents of the eight-bit source register to the destination register. The source remains unchanged.

MOV rdest,M: The HL register pair is assumed to contain a memory address. This instruction moves the contents of that location from memory, into the destination register.

MOV M,rsource: This instruction moves data from the source register into the memory location addressed by the HL register pair.

MVI rdest,bdata: The Move Immediate instruction loads the specified register with the data specified in the instruction.

MVI M,bdata: The Move to Memory Immediate instruction puts the specified data into the memory location pointed to by the HL register pair.

LXI rp,wdata: The Load Register Pair Immediate instruction loads the specified register pair (B, D or H) with the 16-blt word which forms byte 2 and byte 3 of the instruction. Remember the 8080 reverses the order of these bytes, so the least significant byte comes first, then the most significant.

LDA addr (Load Accumulator Direct): This instruction loads the accumulator directly from the address specified.

STA addr: This instruction stores the accumulator contents directly into the address specified.

LHLD addr: Loads the HL register pair directly from the address specified, and the byte following it.

SHLD addr: Stores the contents of the HL register pair directly into memory.

LDAX rp (Load Accumulator Indirect): This instruction specifies either the BC or DE register pairs, which are assumed to contain the address of a location in memory. The contents of this location are moved into the accumulator.

STAX rp: The contents of the accumulator are stored into the memory location pointed to by either the BC or DE register pair.

XCHG: This instruction exchanges the contents of the HL and DE register pairs.

Arithmetic Group

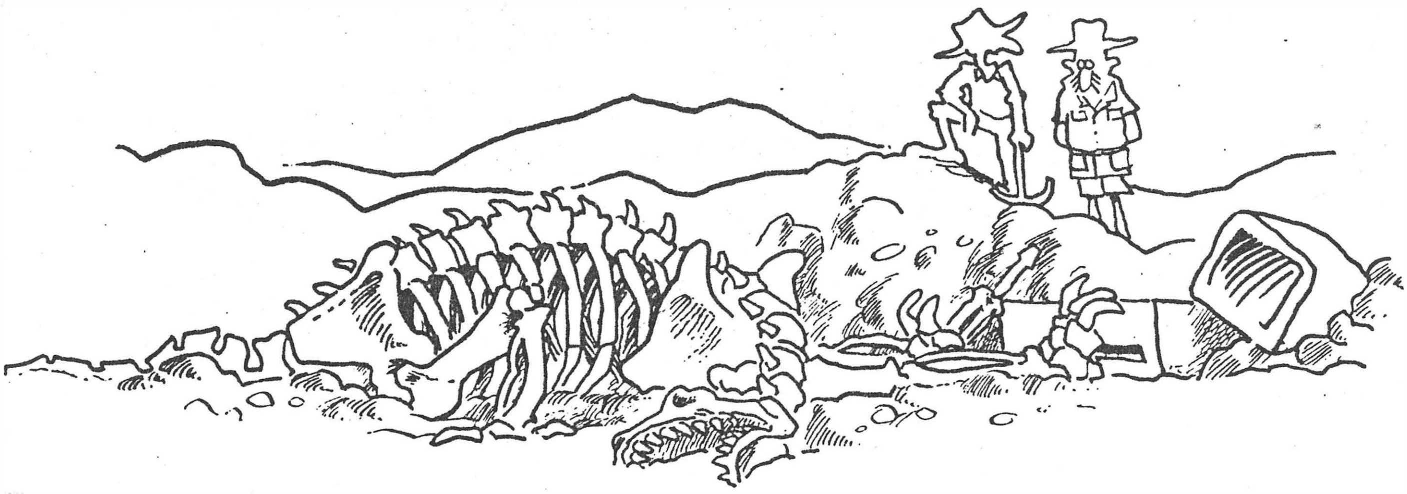
These are the instructions which operate on the accumulator, together with the contents of other registers or memory. In each case the results of the operation are placed in the accumulator and, unless otherwise indicated, the flags are set to reflect the result of the calculation. There are five flags: **Z**ero, **S**ign, **P**arity, **C**arr**Y**, and **A**uxiliary **C**arry.

Subtractions are performed using two's complement arithmetic and set the carry flag to indicate a borrow and clear it to indicate no borrow.

ADD rsource: Adds the contents of the source register to the accumulator.

ADD M: Adds the contents of the memory location pointed to by HL into the

XC Mar 82 1of 4



accumulator.

ADI bdata (Add Immediate): Adds the data specified in the instruction to the contents of the accumulator.

ADC rsource (Add with Carry): Adds the contents of the specified register, plus the carry bit, into the accumulator.

ADC M: Adds the contents of the memory location pointed to by HL, plus the carry bit, into the accumulator.

ACI bdata (Add with Carry Immediate): Adds the data specified in the instruction, plus the carry bit, into the accumulator.

SUB rsource: Subtracts the contents of the specified register from the accumulator.

SUB M: Subtracts the contents of the memory location pointed to by HL from the accumulator.

SUI bdata: Subtracts the data specified in the instruction from the accumulator.

SBB rsource (Subtract with Borrow): Subtracts both the contents of the specified register and the content of the carry flag from the accumulator.

SBB M: Subtracts both the contents of the memory location pointed to by HL and the content of the carry flag from the accumulator.

SBI bdata: The data specified in the instruction and the carry flag are both subtracted from the accumulator.

INR rdest: Increments the contents of the destination register by one. Does *not* affect the carry flag.

INR M: Increments the contents of the memory location pointed to by HL by one.

DCR rdest: Decrements the contents of the destination register by one. Does *not* affect the carry flag.

DCR M: Decrements the contents of the memory location pointed to by HL by one. Does not affect the carry flag.

INX rp: Increments the contents of the specified register pair by one. *No* condition flags are affected.

DCX rp: Decrements the contents of the specified register pair by one. *No* condition flags are affected.

DAD rp (Double-precision Add): Adds the contents of the register pair specified into the HL register pair. *Only the carry flag* is affected. This is a 16-bit addition. Note that DAD H adds HL to HL; that is, it doubles HL.

DAA: Following the addition of two BCD numbers using the ADD or ADC instructions, the result will be incorrect. The DAA instruction converts this result into a valid BCD number.

Logical Group

This group of instructions performs logical operations on registers and memory. Again, the accumulator is involved in all instructions, and the flags are affected, unless noted.

ANA rsource: The content of the specified register is ANDed with the the accumulator. The carry flag is cleared.

ANA M: The contents of the location pointed to by HL is ANDed with the accumulator. The carry flag is cleared.

ANI bdata: The data specified in the instruction is ANDed with the accumulator. The carry and auxiliary carry flags are cleared.

XRA rsource: The contents of the specified register are exclusive-ORed with the acumulator. The carry and auxiliary carry flags are cleared.

XRA M: The contents of the memory location pointed to by HL are exclusive-ORed with the accumulator. The carry and auxiliary carry flags are cleared.

XRI bdata: The data specified in the instruction are exclusive-ORed with the accumulator. The carry and auxiliary carry flags are cleared.

ORA rsource: The contents of the specified register are inclusive-Ored with the accumulator. The carry and auxiliary carry flags are cleared.

ORA M: The contents of the memory location pointed to by HL are inclusive-ORed with the accumulator. The carry and auxiliary carry flags are cleared.

ORI bdata: The data specified in the instruction are inclusive-ORed with the accumulator. The carry and auxiliary carry flags are cleared.

CMP rsource: The flags are set as though the data in the specified register were subtracted from the accumulator, although the accumulator remains unchanged. The Z flag is set if the two registers are equal, and CY flag is set if the accumulator is less than the register.

CMP M: As for CMP rsource, except that HL is used to point to the memory location to be compared.

CPI bdata: The flags are set as though

YC Mar 82 2 of 4

the data specified in the instruction were subtracted from the accumulator, although the accumulator remains unchanged.

RLC: The contents of the accumulator are rotated left one position, and both the carry flag and least significant bit of the result are set to the value shifted out of the most significant bit. Only the carry flag is affected.

RRC: The contents of the accumulator are rotated right one position, and both the carry flag and the most significant bit of the result are set to the value shifted out of the least significant bit. Only the carry flag is affected.

RAL: The accumulator contents are rotated one position left through the carry flag. Only the carry flag is affected.

RAR: The accumulator contents are rotated one position right through the carry flag. Only the carry flag is affected.

CMA: The contents of the accumulator are complemented (1 to 0, 0 to 1). No flags are affected.

CMC: The carry flag is complemented. No other flags are affected.

STC: The carry flag is set to 1. No other flags are affected.

Control Flow Group

These instructions control the sequence of operation of the processor. Jumps may be unconditional or conditional. Unconditional jumps simply load the program counter with the new value, whereas conditional jumps examine the status of the flags to see whether a jump should be performed. The conditions which may be specified are as follows:

NZ Not zero
Z Zero
NC No carry
C Carry
PO Parity odd
PE Parity even
P Plus
M Minus

JMP addr: Unconditional jump to the address specified in the instruction.

Jcond addr: Conditional jump. For example, JPE is jump on parity even, JZ is jump on zero, JNZ is jump on not zero.

CALL addr: Jump unconditionally to the address specified, leaving the return address on the stack. This is the address of the next instruction that would have been executed in the normal course of events.

Ccond addr: Conditional call instruction; example, CZ is call on zero.

RET: Return from subroutine by removing return address from stack and jumping to it.

Rcond: Conditional return.

RST n: Call to one of eight specially defined locations in memory, where the target address is eight times the value of n.

PCHL: Load the program counter with the value in HL.

This group of instructions manipulates the stack, performs I/O and performs other miscellaneous operations.

PUSH rp: Push the specified register pair onto the stack.

POP rp: Pop the the specified register pair off the stack.

PUSH PSW: Push the Program Status Word (that is, accumulator and flags) onto the stack.

POP PSW: Pop the Program Status Word off the stack.

XTHL: Exchange the contents of HL with the two bytes on top of the stack.

SPHL: Load the stack pointer with the value in HL.

IN port: Input a value to the accumulator from the eight-bit port specified.

OUT port: Output the value in the accumulator to the port specified.

EI: Following execution of the next instruction (often a RET) interrupts will be enabled.

DI: Following execution of the next instruction, interrupts will be disabled.

HLT: Halts the processor.

NOP: No operation is performed. This instruction is used to leave space for debugging, and to pad out timing loops.

Assembler Pseudo-Ops

These are instructions to the assembler, and do not generate any code.

ORG addr: Sets the assembler code pointer to generate code starting at this address.

END: Ends a program.

EQU: Sets the value of a label.

SET: Sets the value of a label, and allows it to be changed afterwards (unlike EQU).

DS n: Reserves n bytes of storage space.

DB bdata: Defines a data byte, or string of data bytes.

DW wdata: Defines a data word, reversing the order of the two bytes.

This completes our introduction to the 8080 instruction set. Next month we shall begin writing programs in earnest.

Opcode	Hex	Octal	Decimal
ACI	CE	316	206
ADC A	8F	217	143
ADC B	88	210	136
ADC C	89	211	137
ADC D	8A	212	138
ADC E	8B	213	139
ADC H	8C	214	140
ADC L	8D	215	141
ADC M	8E	216	142
ADD A	87	207	135
ADD B	80	200	128
ADD C	81	201	129
ADD D	82	202	130
ADD E	83	203	131
ADD H	84	204	132
ADD L	85	205	133
ADD M	86	206	134
ADI	C6	306	198
ANA A	A7	247	167
ANA B	A0	240	160
ANA C	A1	241	161
ANA D	A2	242	162
ANA E	A3	243	163
ANA H	A4	244	164
ANA L	A5	245	165
ANA M	A6	246	166
ANI	E6	346	230
CALL	CD	315	205
CC	DC	334	220
CM	FC	374	252
CMA	2F	057	047
CMC	3F	077	063
CMP A	BF	277	191
CMP B	B8	270	184
CMP C	B9	271	185
CMP D	BA	272	186
CMP E	BB	273	187
CMP H	BC	274	188
CMP L	BD	275	189
CMP M	BE	276	190
CNC	D4	324	212
CNZ	C4	304	196
CP	F4	364	244
CPE	EC	354	236
CPI	FE	376	254
CPO	E4	344	228
CZ	CC	314	204
DAA	27	047	039
DAD B	09	011	009
DAD D	19	031	025
DAD H	29	051	041
DAD SP	39	071	057
DCR A	3D	075	061
DCR B	05	005	005
DCR C	0D	015	013
DCR D	15	025	021
DCR E	1D	035	029
DCR H	25	045	037
DCR L	2D	055	045
DCR M	35	065	053
DCX B	0B	013	011
DCX D	1B	033	027
DCX H	2B	053	043
DCX SP	3B	073	059
DI	F3	363	243
EI	FB	373	251
HLT	76	166	118
IN	DB	333	219
INR A	3C	074	060
INR B	04	004	004
INR C	0C	014	012
INR D	14	024	020
INR E	1C	034	028

Opcode	Hex	Octal	Decimal
INR H	24	044	036
INR L	2C	054	044
INR M	34	064	052
INX B	03	003	003
INX D	13	023	019
INX H	23	043	035
INX SP	33	063	051
JC	DA	332	218
JM	FA	372	250
JMP	C3	303	195
JNC	D2	322	210
JNZ	C2	302	194
JP	F2	362	242
JPE	EA	352	234
JPO	E2	342	226
JZ	CA	312	202
LDA	3A	072	058
LDAX B	0A	012	010
LDAX D	1A	032	026
LHLD	2A	052	042
LXI B	01	001	001
LXI D	11	021	017
LXI H	21	041	033
LXI SP	31	061	049
MOV A,A	7F	177	127
MOV A,B	78	170	120
MOV A,C	79	171	121
MOV A,D	7A	172	122
MOV A,E	7B	173	123
MOV A,H	7C	174	124
MOV A,L	7D	175	125
MOV A,M	7E	176	126
MOV B,A	47	107	071
MOV B,B	40	100	064
MOV B,C	41	101	065
MOV B,D	42	102	066
MOV B,E	43	103	067
MOV B,H	44	104	068
MOV B,L	45	105	069
MOV B,M	46	106	070
MOV C,A	4F	117	079
MOV C,B	48	110	072
MOV C,C	49	111	073
MOV C,D	4A	112	074
MOV C,E	4B	113	075
MOV C,H	4C	114	076
MOV C,L	4D	115	077
MOV C,M	4E	116	078
MOV D,A	57	127	087
MOV D,B	50	120	080
MOV D,C	51	121	081
MOV D,D	52	122	082
MOV D,E	53	123	083
MOV D,H	54	124	084
MOV D,L	55	125	085
MOV D,M	56	126	086
MOV E,A	5F	137	095

Opcode	Hex	Octal	Decimal
MOV E,B	58	130	088
MOV E,C	59	131	089
MOV E,D	5A	132	090
MOV E,E	5B	133	091
MOV E,H	5C	134	092
MOV E,L	5D	135	093
MOV E,M	5E	136	094
MOV H,A	67	147	103
MOV H,B	60	140	096
MOV H,C	61	141	097
MOV H,D	62	142	098
MOV H,E	63	143	099
MOV H,H	64	144	100
MOV H,L	65	145	101
MOV H,M	66	146	102
MOV L,A	6F	157	111
MOV L,B	68	150	104
MOV L,C	69	151	105
MOV L,D	6A	152	106
MOV L,E	6B	153	107
MOV L,H	6C	154	108
MOV L,L	6D	155	109
MOV L,M	6E	156	110
MOV M,A	77	167	119
MOV M,B	70	160	112
MOV M,C	71	161	113
MOV M,D	72	162	114
MOV M,E	73	163	115
MOV M,H	74	164	116
MOV M,L	75	165	117
MVI A	3E	076	062
MVI B	06	006	006
MVI C	0E	016	014
MVI D	16	026	022
MVI E	1E	036	030
MVI H	26	046	038
MVI L	2E	056	046
MVI M	36	066	054
NOP	00	000	000
ORA A	B7	267	183
ORA B	B0	260	176
ORA C	B1	261	177
ORA D	B2	262	178
ORA E	B3	263	179
ORA H	B4	264	180
ORA L	B5	265	181
ORA M	B6	266	182
ORI	F6	366	246
OUT	D3	323	211
PCHL	E9	351	233
POP B	C1	301	193
POP D	D1	321	209
POP H	E1	341	225
POP PSW	F1	361	241
PUSH B	C5	305	197
PUSH D	D5	325	213
PUSH H	E5	345	229

Opcode	Hex	Octal	Decimal
PUSH PSW	F5	365	245
RAL	17	027	023
RAR	1F	037	031
RC	D8	330	216
RET	C9	311	201
RLC	07	007	007
RM	F8	370	248
RNC	D0	320	208
RNZ	C0	300	192
RP	F0	360	240
RPE	E8	350	232
RPO	E0	340	224
RRC	0F	017	015
RST 0	C7	307	199
RST 1	CF	317	207
RST 2	D7	327	215
RST 3	DF	337	223
RST 4	E7	347	231
RST 5	EF	357	239
RST 6	F7	367	247
RST 7	FF	377	255
RZ	C8	310	200
SBB A	9F	237	159
SBB B	98	230	152
SBB C	99	231	153
SBB D	9A	232	154
SBB E	9B	233	155
SBB H	9C	234	156
SBB L	9D	235	157
SBB M	9E	236	158
SBI	DE	336	222
SHLD	22	042	034
SPHL	F9	371	249
STA	32	062	050
STAX B	02	002	002
STAX D	12	022	018
STC	37	067	055
SUB A	97	227	151
SUB B	90	220	144
SUB C	91	221	145
SUB D	92	222	146
SUB E	93	223	147
SUB H	94	224	148
SUB L	95	225	149
SUB M	96	226	150
SUI	D6	326	214
XCHG	EB	353	235
XRA A	AF	257	175
XRA B	A8	250	168
XRA C	A9	251	169
XRA D	AA	252	170
XRA E	AB	253	171
XRA H	AC	254	172
XRA L	AD	255	173
XRA M	AE	256	174
XRI	EE	356	238
XTHL	E3	343	227

YC Mar 82 4 of 4

UNDERSTANDING ASSEMBLER

PART III

In last month's article LES BELL introduced the complete instruction set of the 8080 microprocessor: this month he starts programming.

THE CHART at the end of last month's article shows each of the possible opcodes for the 8080 in hex (the preferred counting system), octal (for old fogies like me) and decimal (for those who have no assembler and must POKE programs into memory).

With the aid of this chart we can now start writing useful programs. We'll start with some arithmetic — for no other reason than it's equally useless to everyone, but doesn't require any special hardware.

Languages like Tiny BASIC, tiny c, C and Pascal have an integer data type; sometimes that's all they have. In general, this uses a 16-bit integer expressed in two's complement form, because that's easy to implement on an 8080 (for a complete run-down on two's complement arithmetic, see part two of Binary for Beginners, in *YC* December '81).

First, let's look at addition. Remember the 8080 can use the BC, DE and HL register pairs as 16-bit registers, with the added ability of 16-bit addition, using the HL pair as an accumulator of sorts. The major limitation is that 16-bit arithmetic does not affect the carry and other flags — but as our arithmetic is limited to 16 bits we won't want to carry anyway.

Assume we want to add two 16-bit numbers; how do we go about it? First, we get the two numbers into the HL and DE registers from memory or wherever they were. The details of this procedure depend upon the rest of your program. Then a DAD D (double precision add DE to HL) instruction will add the numbers together, leaving the result in HL. Where the result is moved after that depends upon the rest of the program.

So, our 16-bit add routine looks like this:

A16 DAD D

Written as a complete assembly language file, we have:

```
;       addition example

        org     0100h

a16:    dad     d

        end
```

The first line, as you will remember, is a comment. The org statement tells the assembler to place the machine code at location 0100 hex and onwards, and then comes our 'program'. The next stage — after creating our source code file using ED, WordStar or some other editor — is to assemble it, using ASM or MAC.

The result will be several files; A16.PRN, A16.HEX, and if MAC is used, A16.SYM. The .PRN file shows the resulting object (machine) code against the source code, thus:

```
                        ;       ADDITION EXAMPLE

0100                            ORG     0100H

0100 19         A16:            DAD     D

0101                            END
```

and the symbol table file shows the addresses and values of all labels and symbolic constants:

```
0100 A16
```

The important file produced by the assembler is the .HEX file. It contains an ASCII representation of the machine code, together with information about load addresses and checksums:

```
:0101000019D5
:0000000000
```

Now the program has been assembled, we can go ahead and test it, using CP/M's Dynamic Debugging Tool (DDT). DDT allows us to load programs into memory and execute them one instruction at a time, while examining and changing registers, and so on.

DDT At Work

Figure 1 shows a sample run of DDT and A16.HEX (you'll notice I've called my program ADD.ASM and ADD.HEX). The black marks (yes, those marks like spilled ink) are in fact notes intended to guide you through the session and explain the various DDT commands.

Our addition program (if you can call it that) seems to work, so we can push on to subtraction. Now the 8080 doesn't have a 16-bit subtraction instruction, so we must tackle this differently. Subtraction is done manually, starting at the right and working left, borrowing when appropriate — we can do the same thing here. First we subtract E from L, then we subtract D from H, with a borrow.

Here's the program:

```
;       16-bit subtraction example

        org     0100h

s16:    mov     a,l     ;use the accumulator
        sub     e       ;subtract E from L
        mov     l,a
        mov     a,h
        sbb     d       ;and D from H with a borrow
        mov     h,a     ;(if one was required for L - E)
        ret

        end
```

YC Apr 82 1 of 3

FIG 1

```
A>ddt add.hex          — STARTS DDT AND LOADS ADD.HEX
DDT VERS 2.0
NEXT  PC
0101 0000
-x                     — EXAMINE REGISTERS
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 JMP  0303
-xp  — CHANGE PROGRAM COUNTER
P=0000 0100
-xd    — CHANGE DE
D=0000 0001
-xh  – AND  HL
H=0000 0001                                                    THIS INSTR. WILL BE
-t   — TRACE                                                   EXECUTED NEXT
C0Z0M0E0I0 A=00 B=0000 D=0001 H=0001 S=0100 P=0100 DAD  D*0101
-t
C0Z0M0E0I0 A=00 B=0000 D=0001 H=0002 S=0100 P=0101 ORA  M*0102
-xp
P=0102 00100  ← DO IT AGAIN              ← OUR ANSWER
-xh
H=0002 0041  (HEX = 65 DECIMAL)
-xd
D=0001 009d (HEX = 157 DECIMAL)
-t
C0Z0M1E0I0 A=D3 B=0000 D=009D H=0041 S=0100 P=0100 DAD  D*0101
-t
C0Z0M1E0I0 A=D3 B=0000 D=009D H=00DE S=0100 P=0101 ORA  M*0102
-^C
A>                                   ↑ =222 DEC.
```

FIG_2.

```
A>ddt sub16.hex
DDT VERS 2.0
NEXT  PC
0107 0000
-xp
P=0000 0100
-xh
H=0000 7  }
-xd       }  7-4 SHOULD = 3
D=0000 4  }
-t
C0Z0M0E0I0 A=00 B=0000 D=0004 H=0007 S=0100 P=0100 MOV  A,L*0101
-t
C0Z0M0E0I0 A=07 B=0000 D=0004 H=0007 S=0100 P=0101 SUB  E*0102
-t
C0Z0M0E1I1 A=03 B=0000 D=0004 H=0007 S=0100 P=0102 MOV  L,A*0103
-t
C0Z0M0E1I1 A=03 B=0000 D=0004 H=0003 S=0100 P=0103 MOV  A,H*0104
-t
C0Z0M0E1I1 A=00 B=0000 D=0004 H=0003 S=0100 P=0104 SBB  D*0105
-t
C0Z1M0E1I1 A=00 B=0000 D=0004 H=0003 S=0100 P=0105 MOV  H,A*0106
-t
C0Z1M0E1I1 A=00 B=0000 D=0004 H=0003 S=0100 P=0106 RET  *937D
-^C                                ↑ CORRECT!
A>
A>ddt sub16.hex
DDT VERS 2.0
NEXT  PC
0107 0000
-xp
P=0000 0100
-xh
H=0000 4  }
-xd       }  4 - 7 SHOULD = -3
D=0000 7  }
-t
C0Z0M0E0I0 A=00 B=0000 D=0007 H=0004 S=0100 P=0100 MOV  A,L*0101
-t
C0Z0M0E0I0 A=04 B=0000 D=0007 H=0004 S=0100 P=0101 SUB  E*0102
-t
C1Z0M1E0I0 A=FD B=0000 D=0007 H=0004 S=0100 P=0102 MOV  L,A*0103
-t
C1Z0M1E0I0 A=FD B=0000 D=0007 H=00FD S=0100 P=0103 MOV  A,H*0104
-t
C1Z0M1E0I0 A=00 B=0000 D=0007 H=00FD S=0100 P=0104 SBB  D*0105
-t
C1Z0M1E1I0 A=FF B=0000 D=0007 H=00FD S=0100 P=0105 MOV  H,A*0106
-t
C1Z0M1E1I0 A=FF B=0000 D=0007 H=FFFD S=0100 P=0106 RET  *937D
-^C                                ↑ CORRECT!
A>
A>
```

By assembling this, then using DDT to test it, we can check that it works and see its operation. Take a look at Figure 2. The arrows show the movement of values between the registers.

How does the program cope with negative numbers — fine! In the second part of Figure 2, 4 from 7 leaves FFFD, which is correct, as in two's complement arithmetic FFFD *is* -3. If you don't believe me, add 1 to FFFD, giving FFFE (= -2), add 1 again giving FFFF (= -1) and add 1 again, giving 0000 (= 0).

Go Forth And...

Multiplication on many computers is basically a matter of repeated addition. For example, 9 by 7 is simply 9 added to itself 7 times.

However, remember the good old days 20, 10 or even two years BC (before calculators) when we used to work out long multiplication problems with paper and pencil? We didn't do it that way at all; instead we did it like this:

```
   367
 x 538
 ------
  2936    8 x 367
  1101    3 x 367, shifted one place left
 1835     5 x 367, shifted two places left
 ------
 197446   Total of intermediate calculations
 ======
```

Try one yourself to jog your memory; and take comfort from one L. Bell getting that example wrong the first time (something I wouldn't have noticed without a calculator!).

Notice how the method works. We reduce the problem to single-digit multiplication, which we know how to do from memorized tables. As each successive digit of the multiplier is used to multiply an intermediate result, we shift the answer one more place to the left. Finally, the intermediate results are added up.

Now a binary computer knows how to multiply by a single digit. How? Well, there are only two possible digits, 1 and 0, and 1 times anything is the same thing, while 0 times anything is 0.

Long Multiplication Simplified

Computers are also good at shifting numbers left to right and vice versa; and they can add. Those are all the elements required for a multiplication routine. The only difference between long multiplication on a computer and long multiplication by hand is that with a computer it makes sense to add the intermediate results as they are calculated, rather than waiting until the end of the calculation.

We can write a multiplication algorithm like this:

 YC Apr 82 2 of 3.

```
M1. Set RESULT equal to zero.
                                  --------------------------------
                                  |                              |
                                  V                              |
M2. Is the leftmost digit of the multiplier a 1? If not,         |
    go to step 4.                                                |
                                                                 |
M3. If yes, then RESULT = RESULT + MULTIPLICAND.                 |
                                                                 |
M4. Shift MULTIPLIER one digit right (drops leftmost             |
    digit). If MULTIPLIER is now zero, calculation is            |
    complete.                                                    |
                                                                 |
M5. Shift MULTIPLICAND one digit left (multiplies it by          |
    2). If MULTIPLICAND is now zero, calculation is              |
    complete. Else go to step 2.                                 |
                                  |                              |
                                  V______________________________|
```

This algorithm is fundamentally the same as for long multiplication by hand. In an assembly language version, we will actually build the result in HL, the multiplier will be DE and the multiplicand in BC. In fact to maintain compatibility with our other routines, we will start the routine with the multiplicand in HL; but the first thing the routine does is move HL to BC.

Note, we are multiplying two 16-bit numbers. The result, therefore, could be as large as 32 bits. Why then build the answer in HL, which is a 16-bit register?

The answer is simply that we are performing 16-bit arithmetic and could not use a 32-bit result. Further, we've just run out of registers on the chip, and would have to start fiddling with memory, so the whole thing becomes too complicated. Bear in mind too that multiplication of large numbers could cause overflow, with no error message or other indication.

Routine Notes And Shifts

A few notes about the routine...

The numbers in brackets in the comments refer to the steps of the algorithm above. Note that although the 8080 has two kinds of rotate instruction, we want 16-bit shift routines for this application. Although the routines carry a bit from one byte to the next, they do not carry right around, so they are shifts.

Also keep in mind which instructions affect the carry and zero flags and which do not. Apart from that the routine is reasonably straightforward.

```
        org     0100h

;       16-bit multiplication routine
;       Uses:
;       multiplier in DE
;       multiplicand in HL
;       overwrites BC, A and flags
;
;       Returns result in HL

mult:   mov     b,h     ;copy hl to bc
        mov     c,l
        lxi     h,0     ;set hl to 0 (M1)

m1:     mov     a,e     ;is ls bit a 1
        rrc             ; (M2)
        jnc     m2      ; (M2)
        dad     b       ;if so, add b to result (M3)
m2:     call    sder    ;shift de right (M4)
        rz              ;if de = 0, we're done
        call    sbcl    ;shift bc right (M5)
        rz              ;if bc = 0, we're done
        jmp     m1      ;loop again

;       Shift DE right, setting Z if DE is zero
;       Uses A and flags
sder:   xra     a       ;zero carry flag
        mov     a,d     ;shift left byte first
        rar
        mov     d,a
        mov     a,e     ;then right byte
        rar
        mov     e,a
        ora     d       ;sets Z if D and E zero
        ret

;       Shift BC left, setting Z if BC is zero
;       Uses A and flags
sbcl:   xra     a       ;zero carry flag
        mov     a,c     ;shift right byte first
        ral
        mov     c,a
        mov     a,b     ;then left byte
        ral
        mov     b,a
        ora     c       ;sets Z if C and B zero
        ret

        end
```

p 63 3 of 3

Assemble the routine and test it on your computer using DDT or a similar debugger/monitor. See what happens when large numbers are multiplied. What about negative numbers?

... And Multiplying By Constants

Multiplication by a constant is generally easier to organise. For example, multiplication by 10 can be done by repeated doubling, plus an addition, as 10 = 2 x (1 + 2 x 2). Thus a segment of code to multiply HL by 10 would be:

```
mul10:  mov     e,l
        mov     d,h
        dad     h       ;double HL (x2)
        dad     h       ;and again (x4)
        dad     d       ;add DE (x5)
        dad     h       ;last time (x10)
```

The method for division is broadly similar to manual long divison. It's not just repeated subtractions — the method is a little more sophisticated than that. But in any case, writing a division routine will involve us deeper in the theory of arithmetic than the theory of assembly language, so I don't propose to delve into it here. If there is enough interest we might return to it later.

Next month we'll move on to more general programming techniques: block fill and moves, string searches and so on.

YC Apr 82 3 of 3

UNDERSTANDING ASSEMBLER

PART IV

By popular demand, LES BELL this month diverges slightly to cover programming I/O ports and, in particular, to write a program to allow a computer to communicate with an acoustic coupler.

your computer tutorial

A NUMBER of people have asked me how they would write a program to enable their computer to talk to the Mi-Computer Club Bulletin Board.

Okay, you win... this month I'll set aside my carefully-planned exposition and deal with input/output, with particular reference to serial I/O ports.

As usual, the program will be written to run under CP/M, and assembled using the CP/M assembler. However, exactly the same principles apply to any computer, and where CP/M operating system functions have been used for console I/O, these can usually be replaced with calls to the monitor program of your computer.

And, Or, Um, Not...

Before getting into the program proper, we should spend a little time on formal logic, the only common interest of philosophers and electronic engineers. Here's a simple example:

IF it is a nice day AND I have $5 THEN I will go to the zoo.

There are three simple statements in the above sentence, each of which can be true or false (T or F):

Statement 1: It is a nice day
Statement 2: I have $5
Statement 3: I will go to the zoo

By linking them together with IF, AND and THEN, we are making the truth or falsehood of the third statement depend upon the first two. Both statements 1 and 2 must be true in order for statement 3 to be made true as a result. If statement 1 is false (it's raining) then statement three is false (I won't go to the zoo).

We can tabulate the possibilities (nice/rainy day, have/haven't $5, go/not go) in a **truth table**:

Stat. 1	Stat. 2	Stat. 3
F	F	F
F	T	F
T	F	F
T	T	T

Fig. 1. Truth table for AND function.

Similarly, in a computer, the Ts and Fs can be replaced by 1s and 0s, so the truth table looks like this:

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

This could be stated: If both X and Y are 1, then Z is 1.

Supposing our logical statement said:

IF I have $5 OR I can borrow $5 THEN I will go to the zoo.
then our three simple statements are

Statement 1. I have $5
Statement 2. I can borrow $5
Statement 3. I will go to the zoo

and they are related by IF, OR and THEN. The truth table looks like this:

Stat. 1	Stat. 2	Stat. 3
F	F	F
F	T	T
T	F	T
T	T	T

Fig. 2. Truth table for OR function

In a computer, the truth table would be most simply represented:

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

This can be stated: if either X OR Y is 1, then Z is 1.

There are a couple of other useful logical operators: NOT and XOR (exclusive-OR). The truth table for NOT is

X	Z
0	1
1	0

In other words, Z is NOT X; Z is the inverse of X.

Exclusive OR is related to OR; here's the truth table:

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

In other words, If either X OR Y (but not both) is 1, then Z is 1. This is most useful as a test for equality; if X and Y are the same, then Z is 0. Another use of XOR is as a selective inverter; you'll notice that if X is 0, then Z = Y, but if X is 1, then Z = NOT Y. Finally, later we'll see that XOR is very useful in encrypting data to make it unreadable and hence secure. Now on to I/O.

Input/output Ports

What is an I/O port? Basically, it's an electrical connection (or interface) through which the computer can communicate with the outside world.

Interfaces come in two flavours; parallel and serial. Inside the computer, information is transferred between processor and memory in parallel; that is, on eight parallel wires. A parallel interface really just connects that data to an outside peripheral.

A serial interface is a bit more complex. In this case, a special integrated circuit is normally used, known as a UART (Universal Asynchronous Receiver/transmitter) or ACIA (Asynchronous Communications Interface Adapter) or similar.

This chip has two separate functions. It

YC May 82 1 of 3.

takes serial data in from the outside world, and converts it to parallel to be placed on the parallel data bus of the computer, and it performs the reverse function for transmitted data.

Let's look at how it does this. Serial data consists of a stream of ones and zeros; to make it easy to decode the information, each character is preceded by a start bit and followed by one, one-and-a-half or two stop bits. To assist with error detection, a seven-bit character sometimes has an eighth parity bit added.

The internal circuitry of the UART generally takes care of all these functions automatically. When you send a character (or byte) to the UART, it will automatically add the start, stop and (if required) parity bits. Similarly, on receiving a character, it strips out the start and stop bits, and checks the parity to see if an error has occurred.

Sometimes the UART can get confused about the start and stop bits, and lose track of how many bits of a character it is supposed to have received; this is called a *framing error*, and the UART will have a status bit to indicate this.

Similarly, if the UART receives a character, and you don't read it quickly enough, the next character to be received will over-write it; this causes the UART to signal an *over-run error*.

Generally speaking, the UART is connected to the data bus of your computer via one or more I/O ports. It has several registers, of which the minimum set are the transmit data register, the receive data register and the status register.

To send a character, you check the status register to see if the transmit data register has been emptied; in other words, to see if the UART is ready to accept a character.

While the UART is sending a character, it will show this on a flag in the status register until the transmission is completed, when the flag will change, and a new character can be accepted.

To receive a character, you again check the status register to see if a character has come in. If it has, you read the receive data register, and this has the side-effect of clearing the data-available flag ready for the next character to arrive.

The registers are accessed via the 8080 IN and OUT instructions. If this all sounds terribly complicated, rest assured, it's not that bad in practice!

A Practical Example

Having discussed in general terms how a UART works, let's go on and write a communications program for an actual serial interface.

In this case, it is the Godbout/compupro System Support 1 board which carries, amongst other things, a full serial port. But remember the same techniques, and an almost identical program, can be applied to any computer.

The UART chip used on this board is the Signetics 2651 (also second-sourced by National Semiconductor). This is about the most complex and powerful UART chip around, and has more than the average number of registers.

There are the two data registers for transmit and receive, which are simply written to and read from. There's also the status register mentioned, which indicates the various conditions of operation of the circuit. Rather than list all the status bits, I'll confine our discussion to the bits needed, and ignore the unecessary ones.

Status bit 0 (the least significant bit) when high indicates the UART is ready to accept a character; when low it indicates the UART is busy. This is normally abbreviated TXRDY.

Status bit 1 is RXRDY and there are no prizes for guessing that when high it indicates that a character has been received and is ready to be read from the receive data register.

The remaining bits indicate the various error conditions as well as the state of the Carrier Detect and Data Set Ready lines of the RS-232C serial interface. These do not concern us; interested readers should obtain the 2651 data sheet.

There are three more registers, all of which can be read or written. Two of them, the mode registers, occupy just one I/O port address. This is accomplished by internal logic that allows the user to write the first mode register and then the second. Therefore it is important to write or read both registers, and to consistently deal with Mode Register 1 first.

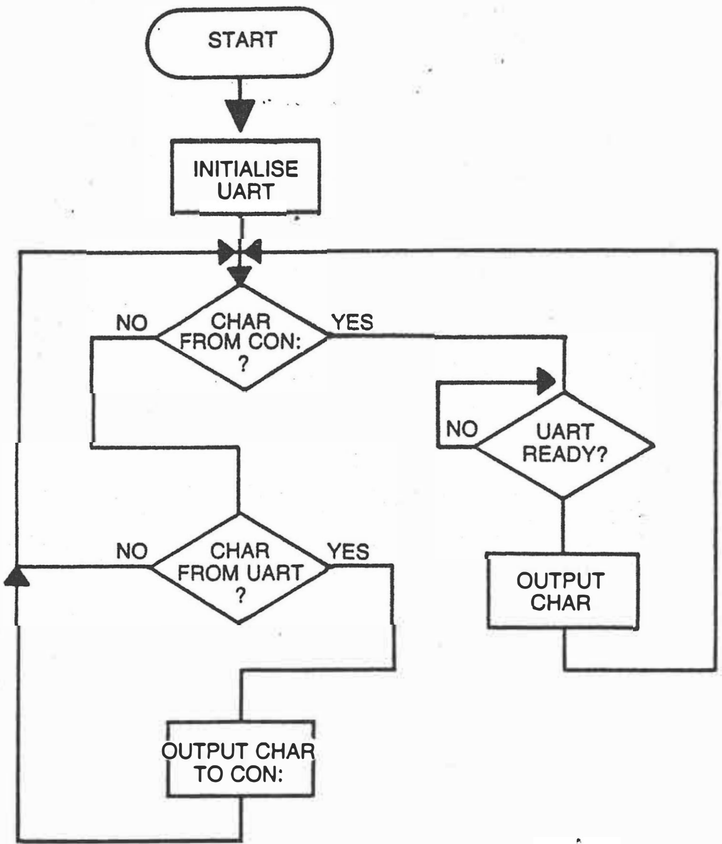
I shan't explain here the detailed operation of the mode registers and command register; but briefly, the mode registers allow the character length, parity, number of stop bits and baud rate (transmission speed) to be set up, while the command register allows control of the RS-232C handshaking lines. These registers normally only have to be set up once, at the beginning of the program; a process that is called initialisation.

With this information, we are now set to write a simple program to make a computer emulate a dumb terminal in order to communicate with a time-share system or bulletin board. We'll start by writing simple routines to input and output a character.

Here's a routine to input a character:

```
rbf     equ     00000010b ;receive buffer full mask

gchar:  in      status    ;check receive status
        ani     rbf       ;received a char?
        jz      gchar     ;no, then wait for it
        in      data      ;then get it
        ret               ;and return
```

How does this work? The first line of code inputs the status byte to the accumu-



YC May 82 2 of 3.

lator. That's fairly straightforward. Now look at the second line. We said above that the receive buffer full flag is bit one of the status byte. In other words if the input status word is

00000010

then a character has been received, while if it is

00000000

there is no character available. That is fine, except that all the other bits of the word can be arbitrarily 1s and 0s, so we have to ignore them. We do this using the ANI (and immediate) instruction. This simply ANDs every bit of the byte following the instruction with the corresponding bit of the accumulator.

If we set up rbf to be 00000010, then the bit we are interested in will remain unchanged, while the others will all be set to 0. Remember, from the discussion above, that X and 0 = 0, while X and 1 = X. Therefore, after the ANI RBF instruction, the accumulator will contain either 00000010 (if a character has been received) or 00000000 (if one has not).

The next step is to jump back and check the status again if the accumulator is zero. Thus, until a character is received, the computer will just loop round and round this bit of code, doing nothing else. On the other hand, if the accumulator contains 00000010, it will not jump, but continue on to input the character.

Simple, isn't it? Transmitting a character is just as easy; here's the code:

```
tbe     equ  00000001b    ;transmit buffer empty mask

        mov  b,a          ;save char in B register
tl:     in   status       ;check status
        ani  tbe          ;transmitter ready?
        jz   tl           ;no, then wait
        mov  a,b          ;get char back
        out  data         ;and output it
        ret
```

In this case, we are going to enter the routine with the character to be transmitted already in the accumulator. However, we are going to use the accumulator to test the status byte, so we temporarily store the character in B, and get it back when we are ready to output it.

If you don't want to use B because it already contains some data from your program, then you can replace the MOV instructions with PUSH PSW and POP PSW respectively, to save the accumulator on the stack. Apart from that, this routine is almost identical to the read routine.

These are the standard routines you would use in a program when you simply want to wait for a character to be input, or output a character with no time restraint. However, our terminal emulator must be able to transmit and receive at virtually the same time. In other words, if a character hasn't arrived, it mustn't wait for one, but check to see if a character is ready to be output. Thus these routines won't work in this application - they need slight modification.

A Terminal Emulator

This terminal emulator must check to see if a character has been input from the computer console keyboard, and send it, if one has. It must also check to see if a character has been received from the modem, and send it to the console if one has.

Input/output under CP/M is done by loading the C register with a function number and then calling location 0005 in memory. In this case, the direct console I/O function is used (function number 6). Here, if E contains FFH, then the function either inputs a character and returns it in A or returns 0 in A if a character was not ready, while if E contains anything else, the character in E is output to the console.

In other systems, such as the TRS-80, MicroBee et al, it is more common to directly call a subroutine in read-only memory (ROM). If your machine has such routines you can ignore the **mvi c,dcio** statements and simply move the character into the appropriate register and call the subroutine directly.

Enough gas-bagging, already! Listing 1 shows the complete program. The first section contains comments and the various equate statements. Notice that although the data registers are at 5CH, the status register is at 5DH and so on, all the registers are specified relative to a single base. This means that should I ever set the address switches on the board to a different address, I only have to change the value of base, and not the rest.

My m1, m2 and c1 are the initialisation words that are written into the mode and command registers of the 2651 integrated circuit by the initialisation routine.

The init routine will probably be different for your computer, if it is needed at all. A previous version of this program, for a different UART, had no initialisation routine.

Four lines after the label *loop:* is the instruction ORA A. You may be wondering what that is doing there; all it does is OR the accumulator with itself, which won't change it. True, it leaves the accumulator unchanged, but it has the side effect of setting the flags according to the accumulator contents.

The preceding call to the BDOS will return zero if no character was available from the console. What we don't know is whether the BDOS's method of putting zero into A will also have set the zero flag.

For example, the instruction MVI A,0 will put zero into A, but does *not* set the zero flag. On the other hand, the instruction XRA A clears the accumulator to zero, and does set the flags. By using the ORA A instruction, the flags are set appropriately for the conditional jump instruction which follows.

We must have some means of exiting the program, so if a character was input, we check to see if it is a control-C. If it is, then we jump back to CP/M (or monitor program).

Apart from these features, the program is fairly straightforward, and corresponds closely to the flowchart. Next month we'll look at block fills (clear screen), block moves and other useful routines. □

```
;       terminal emulator program
;       Written 1/13/82 by Les Bell
;       Modified 3/26/82 to drive Godbout SS1 board
;
base    equ     50h
data    equ     base+0ch        ;2651 data register
status  equ     base+0dh        ;2651 status register
mode    equ     base+0eh        ;2651 mode registers
cmnd    equ     base+0fh        ;2651 command register
;
tbe     equ     1               ;transmit buffer empty flag
rbf     equ     2               ;receive buffer full flag
;
m1      equ     01101110b       ;8 bits,no parity,1 stop bits
m2      equ     01110101b       ;300 baud
c1      equ     00100111b       ;rx and tx enabled, RTS high
;
boot    equ     0000h
bdos    equ     0005h
dcio    equ     6               ;direct console i/o
;
        org     0100h
;
init:   mvi     a,m1            ;reset 8251 mode
        out     mode
        mvi     a,m2
        out     mode            ;mode word 2
        mvi     a,c1
        out     cmnd
;
loop:   mvi     c,06            ;this is where the story. . .
        mvi     e,0ffh          ;input a char
        call    bdos
        ora     a               ;is it really a char?
        jz      gmod            ;no, then check modem
;
        cpi     03h             ;^C?
        jz      boot            ;then reboot
        mov     b,a             ;save char
tl:     in      status          ;then transmit it
        ani     tbe             ;transmitter ready?
        jz      tl              ;no, then wait
        mov     a,b             ;get char back
        out     data            ;output it
        jmp     loop            ;and round again
;
gmod:   in      status          ;check receive status
        ani     rbf             ;received a char?
        jz      loop            ;no, then loop again
        in      data            ;else get it
        mov     e,a             ;and output it
        mvi     c,dcio          ;via the bdos
        call    bdos
        jmp     loop            ;and round again
        end
```


YC May 82 3 of 3

your computer

tutorial

Understanding Assembler —Part V

After a brief digression to talk about communications programs, this month Les returns to more general applications, such as memory fills, block moves and searches...

AS YOU come to write more and more assembly language programs, you will generally find there are certain building blocks that occur again and again.

These include filling a block of memory with a particular byte or pattern, moving a block of memory from one location to another, and searching for a particular word.

Filling memory

Filling memory with a pattern has many uses. For example, many computers, such as the TRS-80, MicroBee and so on have memory-mapped video: that is, the contents of a block of the computer's memory are displayed on the screen. Clearing the screen is a matter of writing spaces into every location of the video RAM.

This is done by loading the character to be written into the accumulator and then using the MOV M,A instruction to repeatedly write it to memory. For example:

```
; Block fill routine
;
        org     0100h

start   equ     0f000h          ;start of block
finish  equ     0f07fh          ;end of block
char    equ     20h             ;char to fill

fill:   lxi     h,start         ;h <- start of block
        lxi     d,finish+1
loop:   mvi     a,char
        mov     m,a             ;store it
        inx     h               ;point to next location
        mov     a,e
        cmp     l
        jnz     loop            ;not equal, keep looping
        mov     a,d
        cmp     h
        jnz     loop            ;not equal, keep looping
        ret
```

When assembled, the code looks like this:

```
                ; BLOCK FILL ROUTINE
                ;
0100                    ORG     0100H

F000 =          START   EQU     0F000H          ;START OF BLOCK
F07F =          FINISH  EQU     0F07FH          ;END OF BLOCK
0020 =          CHAR    EQU     20H             ;CHAR TO FILL

0100 2100F0     FILL:   LXI     H,START         ;H <- START OF B
0103 1180F0             LXI     D,FINISH+1
0106 3E20       LOOP:   MVI     A,CHAR
0108 77                 MOV     M,A             ;STORE IT
0109 23                 INX     H               ;POINT TO NEXT L
010A 7B                 MOV     A,E
010B BD                 CMP     L
010C C20601             JNZ     LOOP            ;NOT EQUAL, KEEP
010F 7A                 MOV     A,D
0110 BC                 CMP     H
0111 C20601             JNZ     LOOP            ;NOT EQUAL, KEEP
0114 C9                 RET

0115                    END
```


YC Jun 82 1 of 3

This version is ORG'ed at 0100H to suit a CP/M system, but others will place it wherever convenient. In practice, however, this routine will probably form part of a larger program and will be called as a subroutine.

An alternative approach is needed when you know the start of the block to be filled, and its length:

```
; Block fill routine version 2
;
        org     0100h

start   equ     0f000h          ;start of block
length  equ     80h             ;length of block
char    equ     20h             ;char to fill

fill:   lxi     h,start         ;h <- start of block
        lxi     d,length
loop:   mvi     a,char
        mov     m,a             ;store it
        inx     h               ;point to next location
        dcx     d               ;decrement de
        mov     a,e
        ora     d
        jnz     loop            ;de not zero, keep looping
        ret
```

Note this version is rather shorter than the first, largely because of the elimination of the compare instructions. In this case, we load DE with the length of the block to be moved, and count down from that value to zero.

The test for zero is accomplished with just two instructions (mov a,e and ora d), which will leave the zero flag set if both d and e are zero.

Block fills will work considerably faster if the transfer takes place to a 256 byte boundary, particularly if the block length is 256 bytes.

In this case, one need only start the fill with HL set to xx00, and increment it until L is 00 again.

```
; Block fill routine, 256 byte block
;
        org     0100h

start   equ     0f000h          ;start of block
char    equ     20h             ;char to fill

fill:   lxi     h,start         ;h <- start of block
loop:   mvi     a,char
        mov     m,a             ;store it
        inx     h               ;point to next location
        mov     a,l
        ora     a
        jnz     loop            ;L not zero, keep looping
        ret
```

Similar techniques can be applied to block fills of other lengths.

Block Moves

Block moves are a little more complex than block fills. In the case of a block move, one usually knows either the beginning and end of the source area, and the beginning of the destination, or the beginning of the source and destination, and the length of the transfer.

For the first case, here's a possible solution:

```
;block mover

        org     0100h

soubeg  equ     0120h           ;beginning of source area
souend  equ     0130h           ;end of source area
dest    equ     0f000h          ;beginning of destination

move:   lxi     h,dest
        lxi     d,soubeg
        lxi     b,souend
loop:   ldax    d               ;get character
        mov     m,a             ;store in memory
        inx     d               ;increment d
        inx     h               ;and h
        mov     a,e             ;compare bc to de
        cmp     c
        jnz     loop
        mov     a,d
        cmp     b
        jnz     loop
        ret
```

This version will move any length of block to any non-overlapping destination.

If moving a 256 byte block of data up one byte, for example, then the block move should start with the last source byte and move it to the end of the destination area, then move downwards through memory from there. A little paper-and-pencil experiment will reveal why.

If the length of the block is known, life can be made a little easier:

```
;block mover, where block length is known

        org     0100h

source  equ     0120h           ;beginning of source area
dest    equ     0f000h          ;beginning of destination
length  equ     20h             ;length of block

move:   lxi     h,dest
        lxi     d,source
        lxi     b,length
loop:   ldax    d               ;get character
        mov     m,a
        inx     d
        inx     h
        dcx     b
        mov     a,c
        ora     b
        jnz     loop
        ret

        org     source
        db      'The quick brown fox jumps over the lazy dog.'
```

This uses the same trick to compare bc for zero as was used in the earlier block fill routine.

A common requirement is to move 128 bytes of data. This is

 YC Jun 82 2 of 3

particularly common in disk operating systems, which deblock longer sectors into 128-byte 'logical sectors' in a buffer, and then have to move the result into the destination area. Here is one way of doing this:

```
;block mover, 128 byte block

        org     0100h

source  equ     0120h           ;beginning of source area
dest    equ     0f000h          ;beginning of destination

move:   lxi     h,dest
        lxi     d,source
        mvi     b,128
loop:   ldax    d               ;get character
        mov     m,a
        inx     d
        inx     h
        dcr     b               ;register decrement sets flags
        jnz     loop
        ret

        org     source
        db      'The quick brown fox jumps over the lazy dog.'
```

These block move routines are one easy way of displaying messages on the screen of a memory-mapped video board, for example. Of course, the exact length of the message must be known in advance, which is rather tedious. A better way of doing this is as follows:

```
; message display routine. Block moves a message until '00'
byte encountered

        org     0100h

pos     equ     0f024h          ;position where message displayed

display:
        lxi     d,msg           ;point to message
        lxi     h,pos           ;point to video display
loop:   ldax    d               ;get char
        ora     a               ;test for zero
        rz                      ;return if zero
        mov     m,a             ;else store in display
        inx     h               ;increment hl
        inx     d               ;and de
        jmp     loop            ;and do again

msg:    db      'The quick brown fox jumps over the lazy dog.'
        db      0
```

This is just a special case of a more general problem, that of finding a particular character.

Without the instructions involving the hl register pair in storing characters to video, this routine could be used to search through a character string looking for a particular character. When the routine returns, hl is pointing to the character.

Typical modifications of this routine would include skipping over a number of letters to find the space that marks the end of a word, or searching for particular characters.

Next month, we shall move on to string searching and pattern matching. YC Jun 82 p 101 3 of 3 □

your computer

tutorial

Understanding Assembler
Part VI

Last month we discussed block fills and moves; the next section of Les Bell's tutorial deals with string output and comparisons. All the examples given in this series are tested and reprinted from the original source code...

ONE OF THE most important functions of any program is output, since a program that doesn't output anything can hardly be said to do anything. In this chapter we are going to look at ways of outputting strings to a printer or video display.

If your computer has a memory-mapped video display, perhaps the quickest way to display a string is to block move it into the display using one of the routines described in the last chapter. If your output device is attached through an I/O port, we shall assume that there is an output routine provided somewhere in the system, either as part of a monitor ROM or as part of the CP/M BIOS, or whatever.

Most systems store strings internally in one of two forms: as a straight sequence of characters, terminated with a zero or other symbol (often '$'); or as a length byte followed by the string with no terminating character.

The first method is generally used for strings which are embedded in programs and do not vary in length, while the second is used for strings held in buffers. In fact, inside CP/M both methods are used: internal messages in the BIOS and transient programs are usually stored in the first way, terminated by a '$' sign, while the CCP (Console Command Processor) stores your command lines in an internal buffer in the second form.

String Encounters

Outputting strings of this kind is fairly easy; it's just a matter of stepping through the string, testing each character for the 'end of string' character, and outputting it if not. Here's how it's done:

```
; routine to output a string via bios calls

        org     0100h
outchr  equ     0d30ch  ; bios console output routine -
                        ; may be different on your sys.
outstr: lxi     h,str
oloop:  mov     a,m
        cpi     '$'
        rz
        mov     c,a
        push    h
        call    outchr
        pop     h
        inx     h
        jmp     oloop

str     db      'The quick brown fox','$'

        end
```

This is represented by the flowchart in Figure 1.

Important points to note: the fetching of each character from memory is performed by the mov a,m instruction. This transfers a byte from the location pointed to by HL into the accumulator.

YC Jul 82 1 of 3

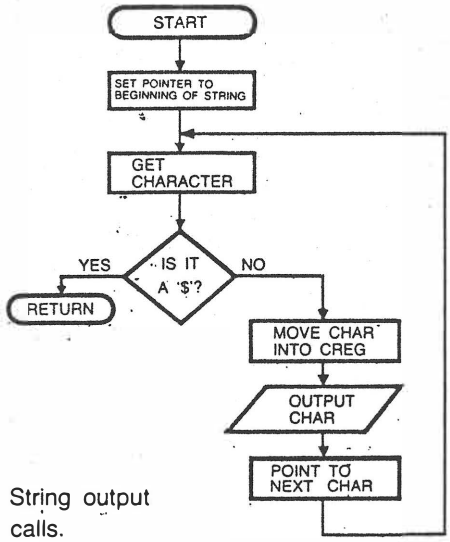


Figure 1. String output via BIOS calls.

This value is the compared with a '$' and the flags set accordingly. If the zero flag is set, then the contents of the accumulator is a dollar symbol, and the routine returns to the calling program, without printing the '$'.

Since HL is used to point to successive characters in the string, it must be preserfed during the bios call. This is done by pushing HL onto the stack and popping it off again after the bios call. HL is then incremented. Note that the bios expects the character to be output to be in the C register, and it is moved there with a mov c,a instruction.

If the string was terminated by a zero byte, then the cpi '$' instruction could be replaced by an ORA A instruction, which would set the flags to reflect the contents of the accumulator. Remember, a MOV instruction does not affect the flags!

Output by means of BDOS calls is slightly different. In this case, C contains the bdos function number, and the character to be output is in E:

```
; routine to output a string via bdos calls

        org     0100h

bdos    equ     0005h   ; bdos jump in page zero
conout  equ     2       ; bdos console out function

outstr: lxi     h,str
oloop:  mov     a,m
        cpi     '$'
        rz
        mov     e,a
        mvi     c,conout
        push    h
        call    bdos
        pop     h
        inx     h
        jmp     oloop
str     db      'The quick brown fox','$'

        end
```

This method has the advantage that the BDOS jump is always at location 5 in all CP/M systems, making for more portable code. It's a little bit slower, though you'd never notice it.

Of course, if you are using BDOS calls under CP/M, remember that there is a function (9) built in for string output, so you don't need a special routine at all:

```
; routine to print a string via bdos call

        org     0100h

bdos    equ     0005h   ;bdos jump in page zero
strout  equ     9       ;bdos string out function

outstr: lxi     d ,str
        mvi     c,strout
        call    bdos
        ret

str     db      'The quick brown fox','$'

        end
```

String Encounters of the Second Kind

In the second kind of string, the first byte of the string contains the string length (excluding the length byte) and the following bytes contain the string itself (see Figure 2).

3	E	O	F	I	L	E

Figure 2. String with length indicator (in this example, the string is 'EOF' not 'EOFILE').

```
; routine to output a string via bios calls

        org     0100h

outchr  equ     0d30ch  ; bios console output routine

outstr: lxi     h,str   ; point to length byte
        mov     b,m     ; get it into b
        inx     h       ; point to first char
oloop:  mov     c,m     ; get char into a
        push    b
        push    h
        call    outchr
        pop     h
        pop     b
        inx     h       ; point to next char
        dcr     b       ; decrement character counter
        rz              ; return if no more chars
        jmp     oloop

str     db      stre - strb
strb    db      'The quick brown fox'
stre:
        end
```

Notice how this works: the first section of code loads the string length into b, which is used as a character counter. As each character is output, b is decremented until it reaches zero, when the last charactep has been output.

Another point is that we no longer have to move the character from memory into a to check its value, but can move it directly into c. Since b is being used as a counter, its value must be

YC Jul 82 2 of 3

preserved during bios calls, so it is pushed on the stack.

Finally, notice how we use the assembler to calculate the string length, rather than doing it manually. By labelling the beginning and end of the string text, we can let the assembler calculate the difference, which is the length.

The program for output via bdos calls is very similar, so we won't go into it here.

Character Searching

Searching for a single character is quite simple. Here's one way of searching through an area of memory, looking for a particular character:

```
; single character search

bdos    equ     0005h   ; bdos jump
spfunc  equ     9       ; bdos string print function

        org     0100h

srch:   lxi     h,start ; load HL with start of search area
        lxi     d,nd    ; load DE with end of search area
loop:   lda     schar   ; load A with search char
        cmp     m       ; compare
        inx     h       ; move to next byte
        jz      found   ; if match, go to found
        mov     a,e     ; get lsb of end address
        sub     l       ; compare with current position
        mov     a,d     ; get msb of end address
        sbb     h       ; subtract
        jm      nfound  ; if less, end of area
        jmp     loop    ; else keep trying
found:  lxi     d,fndmsg
        jmp     print
nfound: lxi     d,nfndmsg
print:  mvi     c,spfunc
        jmp     bdos

start   db      1,2,3,4,5,6,7,8,9
        db      10,11,12,13,14,15,16
nd      equ     $-1

schar   db      11

fndmsg: db      'Byte found$'
nfndmsg db      'Byte not found$'

        end
```

Points to note: the '$' symbol in the equate for nd represents the current value of the assembly pointer, in other words the current address. Thus nd will be set to $-1, the address of the last byte in the area being searched.

The printing of the final message is done by jumping to, not calling, the bdos. The reason for this is that once the message is printed, there is no need to return to the program.

Since we jumped to the bdos, when it returns, it will return not to the program but to the ccp (console command processor), which originally called our program. This will become clearer when we study the stack in the next section.

The best way to understand this program is to step through it using DDT, and changing the value of the byte being searched for so that it will or will not be found.

String Compares

Before we can start searching for strings we must be able to compare them, so that we can tell whether we've found the right string or not.

In this example, I'll take the case of a compiler checking to see whether the word it has come to is a keyword or not.

The compiler will have a table of keywords, each terminated with a null (zero) byte. Thus the comparison consists of checking each letter in turn of the two strings until either they don't match or we reach the zero at the end of the keyword, in which case we have found a match

Here's the code:

```
; match byte sequence with null-terminated word
; pointed to by DE

bdos    equ     0005h
spfunc  equ     9

        org     0100h

        lxi     h,bytes ; input text to scan
        lxi     d,word  ; keyword to compare
loop:   ldax    d       ; examine next letter of keyword
        ora     a       ; is it zero?
        jz      match   ; if yes, we have a match
        cmp     m       ; compare it with text
        inx     d       ; move to next letter
        inx     h       ; on text and word
        jz      loop    ; ok so far, else
        lxi     d,nmmsg ; print no match message
print:  mvi     c,spfunc
        jmp     bdos
match:  lxi     d,matmsg    ; print match message
        jmp     print

nmmsg   db      'No match$'
matmsg  db      'Match found$'

bytes   db      'INPUT A$(1)'   ; typical input text line
word    db      'INPUT',0       ; keyword to compare

        end
```

This is really fairly straightforward. As before, the best way to understand it is to DDT it — and in any case, the experience with DDT will stand you in good stead when debugging your own programs.

Homework time. Notice that in the single character search routine, the test for equAlity is performed by a single instructiOn (cmp m). A string search routine can be written by replacing that instruction with the string compare subroutine.

Bear in mind that these two routines use the HL and DE registers for different purposes, so temporary storage will have to be arranged for values. I'll show one way of doing it next month, but you might like to try it yourself before then.

Next month we'll move on to an investigation of the stack pointer. □

Understanding Assembler

Part VII

As a long-time member of the Society for Putting Things On Top of Other Things, Les Bell has long known about stacks. However, as he explains below, microprocessor stacks are not a quick way to lose things...

IN THE last episode, we developed linear search techniques and a technique for string comparison, and the reader's exercise was to find some way of bringing the two together in a workable way. There are several ways of doing this, but the most useful is a key concept in assembly language programming.

This is to treat program segments as subroutines which can be called from elsewhere within programs, in a manner analogous to the GOSUB in BASIC. A CALL instruction is like a JMP instruction with one important difference; before it jumps, it leaves the address of the next instruction to be executed in a special data area known as the stack.

Stacks are an interesting way of organising data storage areas which offer several advantages over more straightforward techniques such as tables. To understand a stack, it is necessary to review briefly the hardware architecture of the 8080/8085/Z-80 family of microprocessors.

The 8080 has several pairs of registers: accumulator and flags, BC, DE and HL. In addition, there is a program counter, which steps through the program being executed, and the stack pointer.

The stack pointer is a 16-bit register which points to the bottom end of an area in memory called the stack. In general, the stack is situated at the top end of the available RAM memory, and grows downwards as it is filled. The way this works is as follows:

Suppose I have a microcomputer system with 4 Kbytes of memory, and I want to run a program which will use the stack pointer. How is this done?

The stack pointer is first initialised to 1000H, that is, the first byte after the end of memory. To store the contents of a register pair on the stack, I use the PUSH rp (register pair) instruction.

When the processor executes the PUSH instruction, the first thing it does is to decrement the stack pointer, so that now it points to 0FFFH, the last byte of memory. Then it writes the most significant byte of the register pair into this location and decrements the stack pointer again (to 0FFEH). The least significant byte of the pair is then written into this location.

```
:      1000H          :    <--+  Initial position
:---------------------:       :
:                     :       :
:      0FFFH          :    <--:
:      (MSB)          :       :  Stack pointer decrements
:---------------------:       :
:                     :       :
:      0FFEH          :    <--+  to here
:      (LSB)          :
:---------------------:
:                     :
```

Figure 1

Following this, if we want to save another register pair, the stack pointer will be decremented twice more, and the next value will be inserted into the stack under the first. Notice that the stack 'grows' downwards.

The reverse of the PUSH instruction is POP rp. When this is

1 of 3 YC Sep. 82.

executed, the processor first fetches the least significant byte of the word, then increments the stack pointer and fetches the most significant byte. Finally, it increments the stack pointer again, to point to the next element on the stack.

Notice that the number of PUSHes and POPs in a program should match up, in much the same way as the brackets in a mathematical equation should match up (in fact, brackets are analogous to stack operation, as users of Hewlett-Packard pocket calculators will realise).

In fact, in some circumstances, the PUSHes and POPs may deliberately not balance, but in general, and especially for beginners, this is a dangerous practice.

The next thing to notice is that the PUSH instruction does not specify a destination, only a source. Thus, the instruction PUSH D will push the contents of the DE register pair onto the stack, but exactly where is determined only by the contents of the stack pointer, which is itself dependent on the number of previous PUSHes and POPs.

Likewise, the POP instruction has no source; the location on the stack which is read depends purely upon previous activity. A corollary of this is that values must be POPped off the stack in the reverse order from that in which they were PUSHed.

This is not a problem: indeed, it is a considerable advantage of the stack. Incidentally, for this reason you will often see reference to stacks as 'first-in, last-out'.

Suspending Operations

The stack is not only a useful place to store data while it is not needed; it is also a useful place to store the status of a program while the processor temporarily does something else. Take, for example, the execution of a subroutine.

Let's briefly recap on what a subroutine is. Some useful pieces of code occur frequently in a program; for example, a routine to output a character to a terminal might occur fairly frequently. Rather than repeat the code throughout the program, the programmer will write it once, in a general purpose form, in such a way that it can be jumped to from anywhere in a program.

Now the problem with jumping to a routine from a number of different places in a program is that the processor must have some way of finding its way back to where it was before it jumped, so that it can carry on with its job. This is done (you guessed it) by leaving a return address on the stack. It works like this. Consider the following segment of code:

```
START:  LXI     SP,STKTOP
        .
        .                       ; figure something out
        .
        CALL    CHAROUT         ; output result
        MVI     A,B
        .
        .                       ; figure something else
        .
        CALL    CHAROUT         ; output that too
        .
        .                       ; do something else
        .
        JMP     BOOT            ; go back to operating system

CHAROUT:
        IN      STATUS          ; check port status
        .
        .                       ; perform character output
        OUT     DATA
        RET
        END
```

How does this work? The secret is in the CALL and RET (return) instructions, which work in a manner similar to PUSH and POP. When a CALL is encountered, the processor's instruction counter is incremented to fetch the two halves of the 16-bit address which it will jump to. As the final part of this fetch sequence, the PC will be incremented again to point to the next instruction.

At this stage, the most significant half of the program counter is placed on the stack, followed by the least significant half. In other words, rather than PUSHing the contents of a register pair, the processor PUSHes the contents of the program counter.

Finally, the address fetched from the CALL instruction is placed into the program counter, so that the processor continues executing the program there. That completes the operation of the CALL instruction.

The processor will now continue on its merry way, with the address that it should return to safely stored on the stack. Once the subroutine has been concluded, the final instruction in it should be a RET (return). This performs the reverse of the CALL instruction.

Now the processor 'POPs' the return address off the stack and places it in the program counter. Execution will therefore resume with the instruction following the CALL.

Notice that it doesn't matter where in the program the CALL instruction is located; the processor will always leave the correct return address and later return to it properly. Furthermore, subroutine calls can be nested; that is, a subroutine can itself call another subroutine, which in turn can call another subroutine, and so on. Because the return addresses are placed on the stack in order and taken off in the reverse order, everything matches up.

Readers who are familiar with BASIC might like to reflect on the similarities between the CALL instruction and BASIC's GOSUB statement. In fact, the GOSUB and RETURN statements are executed in exactly the same way as the CALL and RET statements, only the user doesn't have to bother about setting up the stack.

Furthermore, BASIC has some nice safeguards built in to protect the user from his own folly if he should, for example, try to RETURN before he's GOSUBbed. What would happen if a machine code program tried to RET without a corresponding CALL.

The answer is that the program would probably go galloping off into the wide blue yonder. In fact, this is one of the most common causes of crashing machine language programs for the novice, and fortunately is one of the easiest to guard against.

Suppose our hypothetical 4K computer has its stack pointer sitting at its initial value, when a RET instruction is encountered.

The processor, not knowing any better, will load the program counter with the contents of non-existent memory; probably FFFFH, or possibly 0000H. As we've agreed there's no memory at FFFFH, the instruction it finds there will be opcode FFH, alias RST 7; this is a call to address 038H, and goodness knows what's located there!

Location 0, on the other hand, will probably be the start of the program, the equivalent of a reset, which, although not exactly desirable, is probably a bit safer than the first case (note to hardware designers: this is a good case for having inverting bus drivers so that any systems which gallop off will fail-safe by resetting. Alternatively, put a jump to an error trapping routine at location 038H).

So, bad stack discipline can be harmful. Let's examine an actual program to show how subroutine calls can be used.

Searching For A Substring

Last month I set readers an exercise of combining substring comparison with a character search routine, to produce a substring search program/routine. I also promised to show one possible solution, so here it is.

In this case, the substring comparison is treated as a subroutine to replace the single-byte comparison in the original character search routine.

2 of 3 YC Sep 82

```
; text search test

bdos    equ     0005h
spfunc  equ     9               ; bdos string print function
wboot   equ     0
        org     0100h

test:   lxi     h,text          ; set up pointers
        lxi     d,ndtext

; search text for substring. Enter with hl set to beginning of
; text and de to end of text

srch:   call    compar          ; compare text with word
        inx     h               ; point to next char
        jz      found           ; if zero, then found
        mov     a,e             ; check for end of text
        sub     l
        mov     a,d
        sbb     h               ; reached end?
        jm      nfound          ; if so, print not found message
        jmp     srch            ; otherwise keep trying
found:  lxi     d,fndmsg
        jmp     print
nfound: lxi     d,nfndmsg
print:  mvi     c,spfunc
        call    bdos
        jmp     wboot
; compar - compare string pointed to by hl with word pointed to by de.
; Return with zero flag set if found.
compar: push    h               ; save text pointers
        push    d
        lxi     d,word          ; point to word
compar1:
        ldax    d               ; get next char of word
        ora     a               ; is it zero?
        jz      goback          ; yes, end of word, found match
        cmp     m               ; compare it with text
        inx     d               ; move to next letter
        inx     h               ; of text and word
        jz      compar1         ; ok so far?
goback: pop     d               ; retrieve original pointers
        pop     h
        ret                     ; and return to caller

; variable areas

text    db      'Now is the time for all good men'
ndtext  equ     $-1
word    db      'tome',0

fndmsg  db      'Word found$'
nfndmsg db      'Word not found$'
        end
```

It's not terribly mysterious. The original string comparison has been written so that instead of printing a found/not found message it sets or resets the zero flag before returning to the calling program.

In almost every respect, the modified search is exactly like the original, and the comparison has not been changed very much either.

Notice that the hl and de register pairs are saved on the stack when the subroutine is entered, as it will use them for its own purposes. Similarly, they must be popped off the stack (in reverse order, of course) before the subroutine is exited.

A little experimenting with DDT will show how the stack pointer goes up and down as registers are PUSHed and POPped.

That's it for this month — next time we'll start looking at the overall design of a monitor program.

YC Sep 82 3 of 3. p59.

your computer tutorial

Understanding Assembler Part VIII

This month, Les Bell looks at the design of a monitor program for an 8080, 8085 or Z-80 based microcomputer.

SO FAR we've looked at routines to perform a number of different functions, including input/output, arithmetic and block moves and searches. It's now time to start putting some of these routines into use, through the design of a simple monitor program.

For those who don't know what a monitor program is, here's a brief description. While many computers these days have BASIC in ROM available on power-up, many systems of more general design have only a bootstrap or in some cases no program at all in memory when switched on. In the old days a front panel consisting of switches and lamps was used to deposit binary instructions into memory and then start the processor executing them.

Generally, the short program keyed in through the front panel was a bootstrap, a short program which would then read in a proper error-detecting loader from paper tape. This loader would then bring in the operating system, BASIC interpreter, or whatever, again from paper tape. A short exposure to this kind of operation soon convinces one of the virtues of floppy disks or even cassette tapes!

Now, front panels are quite expensive, mechanically unreliable in comparison with the rest of the CPU, and add complexity. They have their uses, but much of their job can be done by a monitor program, and so most microcomputer and minicomputer systems today have dispensed with the front panel and replaced it with either a bootstrap ROM which loads the operating system from floppy disk, or a monitor program which loads the system from cassette tape.

The monitor program is a software equivalent to the front panel. It allows the user to 'get inside' the machine, and examine memory locations, change them, start programs running, load programs, save programs, dump parts of memory in either hexadecimal or ASCII, and perform miscellaneous other functions.

System Design

Many of the functions that a monitor performs have already been introduced in this series, and some others we will have to program as we go along.

In any project of this nature, it is best to set down a complete description of your aims and objectives and to think these out quite carefully to avoid conflicts and take note of any compromises that may be necessary.

I should point out at this stage that I am in fact designing this monitor as we go; 'live' so to speak, and it is not already complete and ready to be produced like a rabbit out of a hat.

This way, we may well pursue a couple of blind alleys; start designing one approach to a problem for example, only to decide that is the wrong way to do it and start again a different way.

That is the real world of program design — particularly with assembly language. Many people become discouraged when they find themselves unable to come up with brilliantly structured programs first time; they don't realise that authors of programming textbooks are only showing the last in a long line of programs they have refined, usually over years. They can't do it first time either!

This monitor program is intended for use in the design of I/O interfaces therefore high on the list of priorities is ability to read and write I/O ports.

This is the major facility missing from DDT that I wish it had, and one that is likely to be of most use to many CP/M'ers. It will also fit in nicely with our companion series on logic and interfacing.

Apart from that, the intention is to provide facilities generally comparable to most monitor programs, or indeed DDT. In addition, the monitor will implement an input-output structure similar to that of CP/M, so that programs developed under it can be transported to CP/M without major difficulty.

Machine Dependencies

This raises the major problem of ensuring that the program can be used on as many different machines as possible.

Of course, we are limited to using the 8080/Z-80 family of machines, but the monitor should run on just about any machine based on one of those chips.

For example, it should make no difference whether the host machine uses an external serial terminal or whether it has a built-in screen like the TRS-80. It should also make no difference how how much memory the machine has, be it 16K or 256K.

For this reason, it seems logical to follow the example set by CP/M in having a machine independent portion (supplied by Digital Research) and a machine dependent portion, the BIOS (Basic Input

1 of 2. YC Nov. 82.

Output System, supplied by the user or computer manufacturer).

Our first task, therefore, is to decide on a common standard for input/output code, and it seems reasonable to again follow CP/M, for the simple reason that it provides a pre-defined standard and the experience gained in this will be useful to CP/M users in customising their own systems.

For ease of development under CP/M, initial versions of the monitor will load at address 0100H, but later we will produce a version which moves itself into high memory in order to debug CP/M programs in low memory.

BIOS Functions

The functions implemented in our I/O section will be the non-disk functions of CP/M 2.2, and they will be entered via a jump table as follows:

Hex addr.	Name	Function
**00	BOOT	Cold start entry point
**03	WBOOT	Warm start entry point
**06	CONST	Check console status
**09	CONIN	Console input
**0C	CONOUT	Write character to console
**0F	LIST	Write character to printer
**12	PUNCH	Write character to punch device
**15	READER	Read character from reader device

Some of these functions will be differently implemented from a standard CP/M 2.2 system.

In particular, the PUNCH and READER functions will probably be redefined to work with cassette tape, so that instead of operating on a single character, they will read or write an entire block of data. That remains to be worked out.

Furthermore, the BOOT and WBOOT functions do not have much meaning under a monitor, as the entry points to the monitor will be part of the monitor itself.

The most important parts to note and to start coding are the CONST, CONIN and CONOUT routines, as these will be essential to even a simple monitor. They have to work as follows:

CONST: this routine checks the console status and returns a value in the accumulator. The value is 00H if no character is ready, and FFH if a character is ready.

CONIN: this routine gets a character from the keyboard and returns with it in A. It can either do its own status checking, or it can call CONST, but in either case, it waits until a character is ready and then reads it.

CONOUT: writes the character in register C to the console. It does its own checking of the output status, and once the console is ready to accept the character it writes it out.

All of these routines are not required to preserve the register contents and so may use all the processor registers. The calling program has the responsibility of saving its registers on the stack if necessary.

Typical Code

Here is some typical code for a device using a 2651 UART chip. This chip has two registers which are of most importance here: the status register and the data register. The status register has two bits to indicate the condition of its receive and transmit buffers.

If bit 1 is high, then a character has been received and can be read from the data port. If bit 0 is high, then the transmit buffer is empty, and a character can be sent by writing it to the data port.

This example is fairly straightforward; others may not be so lucky. Owners of memory-mapped video boards, for example, may need to write simple routines to write a character to the screen, or at best, re-write the code supplied with the boards to output from the C register, and then reassemble it.

TRS-80 owners may well be able to find a routine in the machine's ROM to do the job for them; otherwise they will have to write a routine from scratch.

Next month, the monitor code will start with memory dump routines. □

```
        org     0100h

data    equ     00h             ;data port
stat    equ     01h             ;status port
mode    equ     02h             ;mode port
cmmd    equ     03h             ;command port
rbf     equ     00000010b       ;receive buffer full bit
tbe     equ     00000001b       ;transmit buffer empty bit

start:  jmp     init

; this is where the monitor will go

        org     0200h

; JUMP TABLE
init:   jmp     boot
        jmp     wboot
        jmp     const
        jmp     conin
        jmp     conout
        jmp     list
        jmp     punch
        jmp     reader

; boot performs initialization of the 2651 UART and any other
; functions you may need

boot:   mvi     a,11101110b     ;asynchronous, 8 bits no parity 2 stop bits
        out     mode
        mvi     a,01111110b     ;9600 baud
        out     mode
        mvi     a,00100111b     ;set up command port
        out     cmmd
        ret

; console status routine, returns 0 if no char, 0FFH if character avail.

const:  in      stat
        ani     rbf             ; mask rbf bit
        rz                      ; if no data, return with zero in A
        mvi     a,0ffh          ; otherwise put 0ffh in A
        ret                     ; and return

; console input routine

conin:  in      stat            ; get status from UART
        ani     rbf             ; mask rbf bit
        jz      conin           ; wait for character
        in      data            ; get character
        ani     7fh             ; strip high bit
        ret

; console output routine

conout:         in      stat            ; get status from UART
        ani     tbe             ; mask tbe bit
        jz      conout          ; wait for buffer to empty
        mov     a,c             ; move character into A
        out     data            ; and send it
        ret

; other functions dummies for this example

wboot:
list:
punch:
reader:
        ret                     ; return

        end
```

2 of 2 YC Nov 82.

your computer

Understanding Assembler tutorial Part IX

Last month's article covered the basics of I/O for a simple monitor program. This month Les continues with a discussion of memory dumping in hex and ASCII...

THE FIRST requirement from any monitor program is to be able to see into memory, and so this month, we shall look at memory dumping as an exercise.

Most microcomputers use hexadecimal numbering to represent 8 and 16-bit values, and despite the well-known advantages of octal (dig, dig!) I shall bow to the sheer mass of public opinion.

How do you convert a number from its internal binary form, into ASCII using hexadecimal? If you're not careful, this can get awfully confusing.

Remember that, as far as the computer is concerned, everything is binary and not hexadecimal. Hex is purely a convenience for the programmer. When thinking about binary values, the programmer groups them into 4-bit 'nibbles' (half a byte), and then converts them into decimal — only where the decimal system runs out of digits, he starts again with the letter A.

The computer can do the same thing. It can isolate a group of four bits, ready to convert it to hex. Next, it can convert that nibble to an ASCII decimal number. Notice that the digits 0-9 have the ASCII values 30H-39H, so to convert a binary number between 0 and 9 to ASCII, we just add 30H to it.

In fact, we can go ahead and add 30H to any 4-bit value, but we must beware of one problem.

Because the digits 0-9 do not immediately precede the letters A-F in the ASCII code, this means that if the binary value before ASCII conversion was in the range A-F (expressed hexadecimally), then it has been converted to ':', ';', '<' '=', '>' or '?'.

We need to adjust the result if this is the case, and this is done by adding the difference between ':' and 'A' (or A-9-1). The whole process is quite simple, and here's the code to do it:

```
; Output a 4-bit value, contained in the lower
; nibble of A, in hex.
h4:
        ani     0fh             ; mask out unwanted nibble
        adi     '0'             ; convert it to ASCII
        cpi     '9'+1           ; if greater than 9, adjust to > A
        cp      hadj
        call    outchr          ; and print it
        ret
hadj:   adi     'A'-1-'9'       ; make up the difference
        ret
```

The first 'and immediate' reduces the bits in the top half of the byte to zero, as otherwise they will upset our addition.

Then we add an ASCII '0' which, you will recall, has a value of 30H. If, for example, the nibble contained 0, then adding 30H will give a result of 30H, which is an ASCII '0'.

Next we check to make sure that the result is not greater than ASCII '9'. If it is, we add an extra 'fudge factor' to bring it up into the range 'A' to 'F'. This is done by the subroutine hadj:. Finally we call a subroutine called outchr, which prints the character in A.

Okay, now we know how to print a nibble in hex, how do we cope with a byte. Is it more than we can chew (ouch!)?

We just do the same thing twice — once for the high nibble, and then again for the low nibble.

The high nibble and low nibble are swapped (or at least the high nibble is shifted down into the low nibble position), so the high nibble is output by the subroutine we just worked out. Before doing the swap, we push the accumulator onto the stack, and now we pop it off and output the low nibble. The code will therefore look like this:

```
; Output an 8-bit value contained in A, in hex
h8:
        push    psw             ; save for later
        rrc                     ; swap two nibbles
        rrc
        rrc
        rrc
        call    h4              ; and output the first
        pop     psw             ; retrieve value and output second
        call    h4
        ret
```

The four 'rotate right' instructions swap the two nibbles; because there are four of them, they might as well be rotate lefts. The last two instructions (call h4 and ret) are redundant if this routine is placed directly above h4 so that control can simply drop through, as we shall see later.

Finally, a trick we shall often want to do is to output a 16-bit value in hex. Generally, 16-bit quantities are dealt with in the HL register pair; they are usually addresses being used for indirect addressing through HL.

This is done in exactly the same way. A routine h16 splits HL into two bytes, and passes each separately to h8. Very simple.

As an example of how these routines are used, here is a short routine which will locate the BDOS entry point and the location of the BIOS jump table in your system. Note that you cannot do this by simply examining memory under DDT, as it patches the BDOS jump to point to itself (to avoid other programs overwriting DDT).

All the code is fairly standard, but notice that almost every routine calls a subroutine twice. The first time it uses a standard subroutine call, but the second time it is arranged that each routine is immediately above the one it calls, and simply runs into it.

Note that this can only be done if your routine ends like:

Dec 82 YC 93-97

```
        call    foobar          ; redundant
        ret                     ; also redundant

foobar: mvi     a,zot
        .
        ret
```

You can then rely on the ret at the end of the called subroutine to return control to the subroutine (unshown) that made the original call to the subroutine which called foobar.

Purists may well wish to consign this technique to the dirty tricks department.

Here's the program:

```
        title   'BDOS/BIOS Locator V 1.0'

boot    equ     0000h
bdos    equ     0005h
conwr   equ     2               ; BDOS character out function
pstrng  equ     9               ; BDOS print string function
acr     equ     0dh             ; ASCII carriage return
alf     equ     0ah             ;    "   line feed

        org     0100h
main:
        lxi     d,bdmsg         ; print 'BDOS at' message
        mvi     c,pstrng
        call    bdos
        lxi     h,bdos+1        ; point to BDOS jump
        mov     e,m             ; get lower byte
        inx     h               ; then get higher
        mov     d,m
        xchg                    ; and move it into HL
        call    h16             ; and print it.
        mvi     e,acr           ; print CRLF
        mvi     c,conwr
        call    bdos
        mvi     e,alf
        mvi     c,conwr
        call    bdos
        lxi     d,bimsg         ; print 'BIOS at' message
        mvi     c,pstrng
        call    bdos
        lxi     h,boot+1        ; point to warm boot jump
        mov     e,m             ; get lower byte
        inx     h               ; then get higher
        mov     d,m
        xchg                    ; move it into HL
        lxi     d,-3            ; and subtract 3 to point to
        dad     d               ; cold boot at beginning of
                                ; BIOS jump table

; Output a 16-bit value contained in HL, in hex.
h16:
        mov     a,h             ; output first two digits
        call    h8
        mov     a,l             ; then the last two



; Output an 8-bit value contained in A, in hex
h8:
        push    psw             ; save for later
        rrc                     ; swap two nibbles
        rrc
        rrc
        rrc
        call    h4              ; and output the first
        pop     psw             ; retrieve value and output second

; Output a 4-bit value, contained in the lower nibble of A, in hex.
h4:
        ani     0fh             ; mask out unwanted nibble
        adi     '0'             ; convert it to ASCII
        cpi     '9'+1           ; if greater than 9, adjust to > A
        cp      hadj
        call    outchr          ; and print it
        ret
hadj:   adi     'A'-1-'9'       ; make up the difference
        ret

; Output the character in A as ASCII.
outchr:
        push    h
        push    d
        mov     e,a
        mvi     c,conwr
        call    bdos
        pop     d
        pop     h
        ret

bdmsg   db      'BDOS located at: $'
bimsg   db      'BIOS located at: $'
```

What A Dump

The reason we got involved in this whole area of outputting hex in the first place was so we could dump memory, remember?

Now, there are two primary ways we want to look at memory: firstly, as hex bytes, and secondly, as ASCII characters so we can identify text in the middle of our programs (where ideally it shouldn't be, but most compilers are slack about these things).

We want our dump to ideally have both of these side by side, for comparison purposes, and we also want the addresses displayed down the left hand side of the screen.

We want 16 bytes at a time displayed, and we want the line break to occur right on a 16-byte boundary. Anything else? That's enough for starters, anyway.

We'll write a subroutine which is passed two parameters: the start address in HL and the end address in DE. It can simply start dumping and keep incrementing HL until it is the same as DE, then quit.

Or can it? Each 16-byte block of memory has to be dumped twice, once in hex and once in ASCII. The routine must therefore remember the start address of each line being dumped, so that it can go back to it the second time.

Furthermore, if while dumping in hex it discovers the end of the dump, it can't just bundy off, but must repeat that segment in ASCII.

Rather than explain the routine abstractly it's probably better to comment on the listing bit by bit, so here goes:

```
title   'Dump routine V 1.0'
```

I use the CP/M MAC and RMAC assemblers, which allow the user to specify a title to appear at the top of each page. ASM doesn't have this feature; if you're using ASM, ignore this line.

```
boot    equ     0000h
bdos    equ     0005h
conwr   equ     2
```

These are the standard equates I stick at the top of most programs (actually there's a few more but I deleted them). These are pulled in using WordStar, which a) saves me typing and b) avoids errors.

Notice that boot is never referred to in this program, but who cares?

```
acr     equ     0dh
alf     equ     0ah
tab     equ     09h
```

ASCII character equates, absolutely standard. Here comes the actual program.

There's a main body, which sets up DE and HL for testing purposes before the dump routine under test:

```
        org     0100h
main:
        lxi     h,02B3h         ; point to start
        lxi     d,036Ah         ; point to finish
dump:
        push    h               ; save base pointer on stack
        call    h16             ; print initial address
        mvi     a,tab           ; and tab
        call    outchr
```

By this stage, we're under way. The first time through, this section of code may print an address that's not a multiple of 16 (actually 02B3H, in this example), but after that, it will always operate on even boundaries.

Whenever dump is jumped to, we are sitting at the beginning of a line, ready to print an address.

```
dl:     mov     a,m             ; get byte from memory
        call    h8
        mvi     a,' '           ; print a space
        call    outchr
```

This section of code retrieves a byte from memory, prints it, then prints a space. Now we move on to the next byte, but before printing it, we check to make sure that we haven't reached the end:

```
inx     h               ; point to next byte
```

p 94.

```
call    d8              ; have we reached the end?
jm      d2              ; dump remaining ascii
```

Subroutine d8 subtracts hl from de and returns with the sign bit appropriately set. The 'jump on minus' to d2 gets us out of the hex dump loop into the ASCII dump loop. Now we have to check that we haven't reached a multiple of 16. If we haven't, we just keep looping, otherwise we print a space and start dumping ASCII.

```
        mov     a,l
        ani     0fh             ; mask lower bits
        jnz     d1              ; if not zero, keep dumping
d2:     mvi     a,' '           ; else space and dump ascii
        call    outchr
```

Earlier, at the beginning of the line, we pushed HL on the stack. Now we can retrieve it and repeat the dump in ASCII.

The 'ani 7fh' instruction strips off the most significant bit of the character so that it is ordinary ASCII and not graphics. Then we check that it doesn't have a lower value than a space, as that would be a control code and potentially disastrous to our nice neat display. Anything nasty is replaced by a dot.

```
        pop     h               ; get base pointer
d4:     mov     a,m             ; get char from memory
        ani     7fh             ; strip msb
        cpi     ' '             ; if less than space
        cm      d7              ; replace with a dot
        call    outchr          ; output character
        inx     h               ; point to next
```

Once again, we check that we haven't reached the end of the block, and failing that, that we haven't reached the end of a line (that is, address a multiple of 16).

If we have reached the end of a block, the 'return on minus' instruction takes us back to the calling program.

```
        call    d8
        rm
d5:     mov     a,l
        ani     0fh
        jnz     d4
```

If we have reached the end of a line, we output a CR-LF pair and jump round to dump again.

```
d6:     mvi     a,acr
        call    outchr
        mvi     a,alf
        call    outchr
        jmp     dump
```

Here's the subroutines that replace control characters with dots and do the address comparison:

```
d7:     mvi     a,'.'
        ret

d8:                             ; reached end yet?
        mov     a,e
        sub     l
        mov     a,d
        sbb     h
        ret
```

Finally, here are the hex output routines:

```
h16:
        mov     a,h
        call    h8
        mov     a,l
h8:
        push    psw
        rrc
        rrc
        rrc
        rrc
        call    h4
        pop     psw
h4:
        ani     0fh
        adi     '0'
        cpi     '9'+1
        cp      hadj
        call    outchr
        ret
hadj:   adi     'A'-1-'9'
        ret
```

p 96.

3 of 4 YC Dec 82.

```
outchr:
        push    h
        push    d
        mov     e,a
        mvi     c,conwr
        call    bdos
        pop     d
        pop     h
        ret
```

Finally, for ease of typing, here's the whole routine:

```
        title   'Dump routine V 1.0'

boot    equ     0000h
bdos    equ     0005h
conwr   equ     2
pstrng  equ     9

acr     equ     0dh
alf     equ     0ah
tab     equ     09h

        org     0100h
main:
        lxi     h,02B3h         ; point to start
        lxi     d,036Ah         ; point to finish

dump:
        push    h               ; save base pointer on stack
        call    h16             ; print initial address
        mvi     a,tab           ; and tab
        call    outchr
d1:     mov     a,m             ; get byte from memory
        call    h8
        mvi     a,' '           ; print a space
        call    outchr
        inx     h               ; point to next byte
        call    d8              ; have we reached the end?
        jm      d2              ; dump remaining ascii
        mov     a,l
        ani     0fh             ; mask lower bits
        jnz     d1              ; if not zero, keep dumping
d2:     mvi     a,' '           ; else space and dump ascii
        call    outchr
        pop     h               ; get base pointer
d4:     mov     a,m             ; get char from memory
        ani     7fh             ; strip msb
        cpi     ' '             ; if less than space
        cm      d7              ; replace with a dot
        call    outchr          ; output character
        inx     h               ; point to next
        call    d8
        rm
d5:     mov     a,l
        ani     0fh
        jnz     d4
d6:     mvi     a,acr
        call    outchr
        mvi     a,alf
        call    outchr
        jmp     dump


d7:     mvi     a,'.'
        ret

d8:                             ; reached end yet?
        mov     a,e
        sub     l
        mov     a,d
        sbb     h
        ret

h16:
        mov     a,h
        call    h8
        mov     a,l
h8:
        push    psw
        rrc
        rrc
        rrc
        rrc
        call    h4
        pop     psw
h4:
        ani     0fh
        adi     '0'
        cpi     '9'+1
        cp      hadj
        call    outchr
        ret
hadj:   adi     'A'-1-'9'
        ret

outchr:
        push    h
        push    d
        mov     e,a
        mvi     c,conwr
        call    bdos
        pop     d
        pop     h
        ret
```

YC Dec 82
4 of 4

Understanding Assembler Part X

Last month, Les Bell developed a hex dump routine, the first function of a monitor program. This month, he continues development with input/output routines, a command interpreter and incorporation of the dump routine...

NOW THAT we have a working hex dump routine, we can set about constructing a monitor proper.

As mentioned previously, this will perform some of the functions performed by CP/M's BDOS, so that programs written to run under it will be transportable to CP/M. Furthermore, following the CP/M practice of having a separate BIOS (Basic Input/Output System) which varies from machine to machine will allow us to simplify transportation between machines.

This month, there's very little theory to follow, just code to examine, so let's get into it...

As always, the first part of the program defines the constants which will be used — in this case mostly ASCII characters which will be used to edit input and control output. We also define the prompt character and the location to jump to when exiting the monitor, in this case, the CP/M warm boot location.

```
; 8080 / Z-80 monitor routines
;

ctrc    equ     03h        ; control-C for abort
acr     equ     0dh        ; carriage return
alf     equ     0ah        ; line feed
ctrh    equ     08h        ; backspace character
tab     equ     09h
ctrs    equ     13h        ; control-S for pause output
ctrx    equ     18h        ; control-X erases input line
clear   equ     1ah        ; TVI 910 clear screen cmd

del     equ     7fh

prompt  equ     '>'
buflen  equ     128
exit    equ     0000h
```

Now here's the program itself. We start with the initialisation routines which first load the stack and them jump to the machine dependent part in the fake BIOS. This does initialisation of hardware.

```
        title   'monitor rev 1.0

        org     0100h

monitor:
        lxi     sp,stk
        call    boot
```

On return from the initialisation we enter the monitor proper. This consists of a simple loop which moves to a new line, puts out the prompt, inputs a line from the terminal and then interprets that line.

```
monl:   call    crlf
        mvi     c,prompt
        call    putch
        lxi     h,buff
        call    getln
        lda     buff
        cpi     acr
        cnz     scanner
        jmp     monl
```

This is an unusual approach for a monitor program: rather than input a line and interpret it, most simply input a character and then jump immediately to the appropriate routine.

I have chosen this approach because it is more general, and it can be used to interpret multiple lines of commands in a buffer, rather like a program. In this respect, this monitor is more like a PILOT interpreter than a monitor.

The advantages are that we have more general code which you can re-use in your own programs, and which can be easily expanded into an interpreter for a simple language. The disadvantage is that the monitor will be rather larger than normal.

YC Jan/Feb. 83.

1 of 4.

Here's the routine which gets a line into the buffer for later interpretation. It is reasonably sophisticated, and allows simple line editing: in other words the backspace key works, and a control-X will delete the line, returning the cursor to the beginning of the line.

It also performs automatic case folding (though this is easily disabled for more general applications), and will automatically return to the calling routine when the 128-byte buffer is full.

```
getln:  mvi     e,0         ; character counter
getlnl: call    conin       ; get a character
        cpi     ' '         ; is it a control char?
        jc      getln4      ; yes, jump to handler
        cpi     'Z'+1       ; is it lower case?
        jc      getln2      ; no, carry on regardless
        ani     5fh         ; otherwise fix it
getln2: mov     m,a         ; store char in buffer
        mvi     a,buflen    ; get buffer length
        cmp     e           ; have we reached it?
        rz                  ; yes, return to caller
        mov     a,m         ; retrieve character
        inx     h           ; bump buffer pointer
        inr     e           ; and counter
        mov     c,a
        call    putch       ; echo char
        jmp     getlnl

getln4: cpi     ctrh        ; is it a backspace?
        jz      getln5
        cpi     ctrx        ; is it a control-X?
        jz      clline
        cpi     acr         ; or a CR?
        jnz     getlnl      ; if not, ignore it.
        mov     m,a         ; otherwise, emit CRLF and return

crlf:   mvi     c,acr
        call    putch
        mvi     c,alf
        call    putch
        ret

getln5:                     ; control-H (backspace) handler
        mov     a,e
        ora     a
        jz      getlnl
        mvi     c,ctrh      ; print backspace
        call    conout
        mvi     c,' '       ; then a space
        call    conout
        mvi     c,ctrh      ; then another backspace
        call    conout
        dcr     e           ; count down
        dcx     h           ; back up buffer pointer
   jmp      getlnl     ; get next character

clline:                     ; control-X (clear line) handler
        mvi     c,ctrh      ; print backspace


cllinel:
        mov     a,e
        ora     a
        jz      getln
        call    conout
        mvi     c,' '       ; print space
        call    conout
        mvi     c,ctrh      ; print backspace
        call    conout
        dcx     h           ; back up buffer pointer,
        dcr     e           ; count back a char
        jnz     cllinel     ; to beginning of line
        jmp     getln       ; and start all over
```

A special character output routine is used which also checks the input status.

If a character is present it is read, and if it is a control-S, output is paused, otherwise output aborts. Once paused, any character will restart output except control-C, which will abort it.

```
putch:
        call    conout
        call    const       ; test for input character
        ora     a           ; set flags to test for zero
        rz                  ; no character, so return
        call    conin
        cpi     ctrs        ; control-S?
        jz      pawz        ; yes, pause
        jmp     monl        ; no, halt
pawz:   call    conin
        cpi     ctrc        ; control-C?
        jz      monl        ; yes, abort
        ret
```

Now we come to the heart of the monitor, a simple interpreter.

This simply starts at the beginning of the line and reads the first character, which must be a single letter command. If it's not, an error is flagged. Once the command has been identified, control is passed to the appropriate routine.

This is done by reading the address from a table (in alphabetical order) and then jumping to that address by using the PCHL (load program counter from HL) instruction.

This is the same basic principle as used by high level language interpreters in executing 'tokenised' languages such as BASIC.

You'll notice that almost all the entries in the table point to the error handling routine. Only the D (dump) and E (exit) commands have been implemented, but more will follow.

```
************************************************************
*                                                          *
*                     simple scanner                       *
*                                                          *
************************************************************

scanner:
        lxi     h,buff
        mov     a,m             ; get a character
        inx     h               ; point to next
        shld    cursor          ; and save cursor
        sui     'A'             ; convert to table index
        jc      error           ; if less than A then error
        cpi     'Z' - 'A' + 1   ; if > 'Z' then error
        jnc     error
        add     a               ; double A
        lxi     h,table         ; point to start of table
        mvi     d,0
        mov     e,a             ; put offset in DE
        dad     d               ; and add it to HL
        mov     e,m             ; get low byte
        inx     h
        mov     d,m             ; get high byte
        xchg                    ; get it into HL
        pchl

table:
        dw      error           ; A
        dw      error           ; B
        dw      error           ; C
        dw      dump            ; D
        dw      exit            ; E
        dw      error           ; F
        dw      error           ; G
        dw      error           ; H
   dw       error           ; I
        dw      error           ; J
        dw      error           ; K
        dw      error           ; L
        dw      error           ; M
        dw      error           ; N
        dw      error           ; O
        dw      error           ; P
        dw      error           ; Q
        dw      error           ; R
        dw      error           ; S
        dw      error           ; T
        dw      error           ; U
        dw      error           ; V
        dw      error           ; W
        dw      error           ; X
        dw      error           ; Y
        dw      error           ; Z

************************************************************
*                                                          *
*                     error handler                        *
*                                                          *
************************************************************

error:
        mvi     c,'?'
        call    conout
        call    crlf
        jmp     monl
```

To match this simple scanner there are several routines which will be used by the command routines to interpret the command line in the buffer. The first of these skips over spaces

2 of 4 YC Jan/Feb 83

and delimiters until it finds a meaningful character.

Note that it regards a carriage return as an error, and should therefore only be called when an argument is expected. Thus a missing argument will be correctly flagged as an error.

```
*************************************************************
*                                                           *
*                      scanner tools                        *
*                                                           *
*************************************************************

spskip:                         ; skip over spaces and delimiters
        lhld    cursor
spskl:  mov     a,m
        inx     h
        cpi     acr             ; no cr expected, error
        jz      error
        cpi     ' '
        jz      spskl
        cpi     ','
        jz      spskl
        cpi     tab
        jz      spskl
        dcx     h
        shld    cursor
        ret
```

The getparm routine reads a string of ASCII digits from the buffer and converts them into a 16-bit binary value in HL. It returns when it runs out of digits.

```
getparm:                        ; get hex parameter
        push    d
        push    b
        lxi     h,0
gpl:    call    getchar
        call    gp2
        jc      gp3             ; if carry set, we're done
        dad     h               ; multiply HL by 16
        dad     h
        dad     h
        dad     h
        ora     l               ; and add A to HL
        mov     l,a
        jmp     gpl

gp2:

; convert ASCII to binary, return with carry set if
; not a valid digit.

        sui     '0'             ; remove ASCII offset
        rc                      ;  character < 0
        cpi     'F'-'0'+1
        cmc                     ; complement carry
        rc                      ; character > F
        cpi     10
        cmc                     ; complement carry
        rnc                     ; ok, number 0 - 9
        sui     'A'-'9'-1
        ret

gp3:                            ; character not hex
        call    ungetch         ; put it back for next routine
        pop     b               ; clean up the stack
        pop     d
        ret
```

Getchar and ungetch are loosely modelled on the C language library calls.

The function of getchar is to get a character from the buffer, advancing the cursor as it goes, while ungetch does the reverse (by the time a routine knows a character is no use to it, it has read it, and must replace it for the next routine).

```
getchar:                        ; get character from buffer
        push    h               ; save HL
        lhld    cursor          ; get cursor
        mov     a,m             ; get character
        inx     h               ; point to next
        shld    cursor          ; save cursor
        pop     h               ; restore HL
        ret

ungetch:                        ; back up cursor
        push    h               ; save HL
        lhld    cursor          ; get cursor
        dcx     h               ; move back
        shld    cursor          ; save cursor
        pop     h               ; restore HL
        ret
```

This dump routine is virtually identical to last month's, except for the code at the beginning which gets the dump addresses into DE and HL by reading the buffer line.

Also, notice that output is done by calls to putch with the output character in the C register.

```
*************************************************************
*                                                           *
*                           dump                            *
*                                                           *
*************************************************************

dump:
        call    spskip
        call    getparm         ; get start address
        xchg                    ; put start into DE
        call    spskip
        call    getparm         ; get finish
        xchg                    ; DE <- finish, HL <- start
d0:     push    h               ; save base pointer on stack
        call    h16             ; print initial address
        mvi     c,tab           ; and tab
        call    putch
d1:     mov     a,m             ; get byte from memory
        call    h8
        mvi     c,' '           ; print a space
        call    putch
        inx     h               ; point to next byte
        call    d8              ; have we reached the end?
        jc      d2              ; dump remaining ascii
        mov     a,l
        ani     0fh             ; mask lower bits
        jnz     d1              ; if not zero, keep dumping
d2:     mvi     c,' '           ; else space and keep dumping
        call    putch
        pop     h               ; get base pointer
d4:     mov     a,m             ; get char from memory
        ani     7fh             ; strip msb
        cpi     ' '             ; if less than space
        cm      d7              ; replace with a dot
        mov     c,a             ; move into C
        call    putch           ; and print it
        inx     h               ; point to next
        call    d8
        jc      crlf            ; exit with a CRLF
d5:     mov     a,l
        ani     0fh
        jnz     d4
        call    crlf
        jmp     d0

d7:     mvi     a,'.'
        ret

d8:                             ; reached end yet?
        mov     a,e
        sub     l
        mov     a,d
        sbb     h
        ret

*************************************************************
*                                                           *
*                  hex output routines                      *
*                                                           *
*************************************************************

h16:
        mov     a,h
        call    h8
        mov     a,l
h8:
        push    psw
        rrc
        rrc
        rrc
        rrc
        call    h4
        pop     psw
h4:
        ani     0fh
        adi     '0'
        cpi     '9'+1
        cp      hadj
        mov     c,a
        call    putch
        ret
hadj:   adi     'A'-1-'9'
        ret
```


3 of 4   YC   Jan/Feb 83

This fake 'BIOS' contains the hardware dependent input/output code for the particular machine used.

You will need to re-write it for your machine; the only thing that must stay the same is the jump table at the beginning. Although the jump table is not used by routines inside the monitor, it will be used by application programs.

```
*************************************************************
*                                                           *
*                   fake bios                               *
*                                                           *
*************************************************************

data    equ     0e3f8h      ;data port
stat    equ     0e3f9h      ;status port
rbf     equ     00000100b   ;receive buffer full bit
tbe     equ     00001000b   ;transmit buffer empty bit

pio     equ     20h         ; parallel I/O chip (8255A)
lstat   equ     pio+2
ldata   equ     pio+1
pcont   equ     pio+3

; JUMP TABLE
fbios:  jmp     boot
        jmp     wboot
        jmp     const
        jmp     conin
        jmp     conout
        jmp     list
        jmp     punch
        jmp     reader
```

The boot routine performs hardware initialisation; in this case, setting up the registers of an 8255 parallel I/O chip to act as a Centronics printer driver. Later, we will add more functions to this routine.

```
boot:   mvi     a,10101100b  ; set 8255 mode
        out     pcont
 mvi    a,00000101b  ; set group b mode
        out     lstat
        mvi     c,clear
        call    conout
        ret

wboot:  jmp     mon1
```

The console status routine is, obviously, highly hardware dependent.

If you are using a serial terminal with a UART chip, your const routine will look somewhat similar to this; if you have a machine with an integrated keyboard, *a la* Apple and Tandy, it will be a call to the input routine together with some code to return the right result in A.

The same comments apply to the console input and output routines. If you are using a UART chip, note that you shouldn't need the CMA instructions, which are used in my system because the UART supplies inverted data onto the bus, a hardware peculiarity of my system.

```
; console status routine, returns 0 if no char,
; 0FFH if character avail.

const:  lda     stat
        ani     rbf       ; mask rbf bit
        mvi     a,0       ; if no data, return with zero in A
        rnz
        dcr     a         ; otherwise put 0ffh in A
        ret               ; and return

; console input routine

conin:  lda     stat      ; get status from UART
        ani     rbf       ; mask rbf bit
        jnz     conin     ; wait for character
        lda     data      ; get character
        cma               ; adjust for negative bus
        ani     7fh       ; strip high bit
        ret

; console output routine

conout: lda     stat      ; get status from UART
        ani     tbe       ; mask tbe bit
        jnz     conout    ; wait for buffer to empty
        mov     a,c       ; move character into A
        cma               ; adjust for negative bus
        sta     data      ; and send it
        cma               ; make positive again
        ret


list:
        in      lstat     ; get lp status
        ani     2
        jz      list
        mov     a,c
        out     ldata
        ret

; other functions dummies for this example

punch:
reader:
        ret               ; return

buff    ds      128
        ds      64
stk     equ     $
cursor  ds      2

        end
```

That completes the first version of the monitor. Next month, we'll go on to add more functions, such as changing memory locations and running a program. □

p55

your computer
tutorial

Understanding Assembler

Part XI

Continuing with the development of a general-purpose monitor program, Les Bell this month looks at examining and changing memory and running a program under the monitor.

LAST MONTH WE got the basic monitor going, with the ability to dump memory to the display as well as exiting back to CP/M, under which the monitor is being developed. This month, we shall move on to a method of changing memory and of jumping to subroutines.

The first problem we can solve in much the same way as DDT uses to examine and change memory: the S (for substitute) command. The user types S, followed by the address at which substitution is to start, and the monitor responds with the contents of that location, then waits for a new value to be input.

If no value is entered, the old contents of the memory location are left unchanged, and the machine goes on to examine the next location. Data input is terminated by input of a full stop.

We have already written many of the subroutines which will be necessary for this job. For example, we already have a line input routine with primitive editing, which will allow us to enter values, and we have routines to input a hex value from the line and to output a hex value to the console.

Writing the substitute command, therefore, should simply be a matter of calling these subroutines in the right order, with a bit of loop control tacked on.

And so it turns out. Our first job, when we enter the routine, is to parse the rest of the command line, to get the address at which the substitute command will start operation. The scanner will already have positioned the cursor just after the letter S of the command line, and so we must first skip spaces to get to the first non-blank character, and then get that number into the HL register pair:

```
subst:
        call    spskip          ; skip over spaces
        call    getparm         ; get start address
```

There's nothing very complex about that. Next we must output the address, followed by a space, then get the contents of that memory location, and print that, followed by another space.

Since we will have to do some hex conversion work in HL, we save the address pointer on the stack for later:

```
substl:
        call    h16             ; output address
        mvi     c,' '           ; and then a space
        call    putch
        push    h               ; save address on stack
        mov     a,m             ; get memory contents
        call    h8              ; output byte in A
        mvi     c,' '           ; space in C
        call    putch           ; and output it
```

Now we need to get a hex number from the keyboard. We just point to the line input buffer and call the *getln* routine:

```
        lxi     h,buff          ; point to buffer
        call    getln           ; get a hex number (or whatever)
```

Having got either a hex number, or a decimal point, or nothing, into the buffer, we now have to examine it and figure out what to do.

If it is a dot, we jump back to the mainline, and if it's a carriage

1 of 2.

YC Mar. 83.

return, we skip over the code which changes the memory location, continuing round the loop.

```
        lxi     h,buff          ; point to beginning of buffer
        shld    cursor          ; set cursor there
        mov     a,m             ; get first char
        cpi     '.'             ; is it a dot?
        jz      monl            ; back to mainline
        cpi     acr             ; if it's a cr, don't change
        jz      subst2          ; skip over change code
```

Now we've worked out that it's not a special character, the rest of the line should contain the hex value to be placed in memory. So the first thing we do is get the value into the HL register pair, using the getparm routine written before, and then move the lower half of that sixteen-bit value into the accumulator.

A POP H instruction will now restore the pointer to memory, and we can store the accumulator into memory using a MOV M,A instruction. The loop is completed with an increment pointer instruction and a jump, and that completes the major part of the code.

```
        call    getparm         ; get hex number
        mov     a,l             ; mov byte into a
        pop     h               ; get back address
        mov     m,a             ; and store byte
        inx     h               ; move on to next
        jmp     substl          ; and loop round
```

The only thing that remains to be taken care of is the code to balance the stack and increment the pointer without storing to memory in response to a carriage return:

```
subst2:
        pop     h               ; restore stack
        inx     h               ; move on
        jmp     substl          ; and loop round
```

That completes the 'substitute' command code. To implement it as part of the monitor, you should use a text editor to read it into the monitor source code file (monitor.asm) and then reassemble the monitor.

You will need to change the entry for 'S' in the scanner jump table to read 'dw subst', so that when the monitor is reassembled, the S command will jump to the routine.

Jumping to Programs

One of the major reasons for using a machine code monitor is the ability to develop routines and programs. For this reason, we need to have the ability to transfer control to a program and return to the monitor upon completion.

Ideally, we should also like to set breakpoints, that is, points at which the program will stop running and hand control back to the monitor so we can see what is happening.

However, let's not try running before we can walk. We'll get to breakpoints in due course; meanwhile, we'll add a 'Go' command to the monitor which will allow us to jump to locations in memory and then return control to the monitor.

The first thing to notice about the go command is that we want to leave a return address on the stack: either the entry point of the monitor or the address of a routine (which can directly follow) which will manage the return. In this case, I've chosen to return to the monitor entry point.

The technique is very simple. We know, upon entry, that the remainder of the command line should contain the address to jump to, so we use the existing spskip and getparm routines to get that address into HL.

We then save that address in DE (using the xchg instruction) and load HL with the address of the warm boot routine. After pushing this onto the stack, ready to act as the destination of a return instruction, we exchange the DE and HL registers again, and load the program counter with the contents of HL. That's it.

```
**********************************************************************
*                                                                    *
*                           go command                               *
*                                                                    *
**********************************************************************

; go (jump) to an address from command line

go:     call    spskip          ; skip over spaces
        call    getparm         ; get address to go to
        xchg                    ; save in DE
        lxi     h,wboot         ; get return address
        push    h               ; and place it on stack
        xchg                    ; get go address again
        pchl                    ; and go
```

The program under development will now run normally (or abnormally – it's under development, after all), and will terminate by performing a RET instruction which will bring it back to the warm boot entry point of the monitor.

Now, this routine can be incorporated into the monitor in exactly the same way, using a text editor to read it in and changing the 'G' entry in the scanner jump table. However, there is one other thing that will have to be changed in the monitor program.

As it stands, the mainline transfers control to the scanner routine through a CALL instruction, pushing a return address onto the stack. Now, if this routine re-enters the monitor by a straight jump to the warm boot entry, as things stand that return address will never get popped off the stack, which will grow down in memory until it overflows. Not good.

The monitor will have to be modified slightly so that the warm boot routine sets up the stack, and not the first entry in the mainline code, as it currently is set up. This is left as an exercise for the reader; next time we print the monitor in its entirety, you'll see how I've chosen to do it.

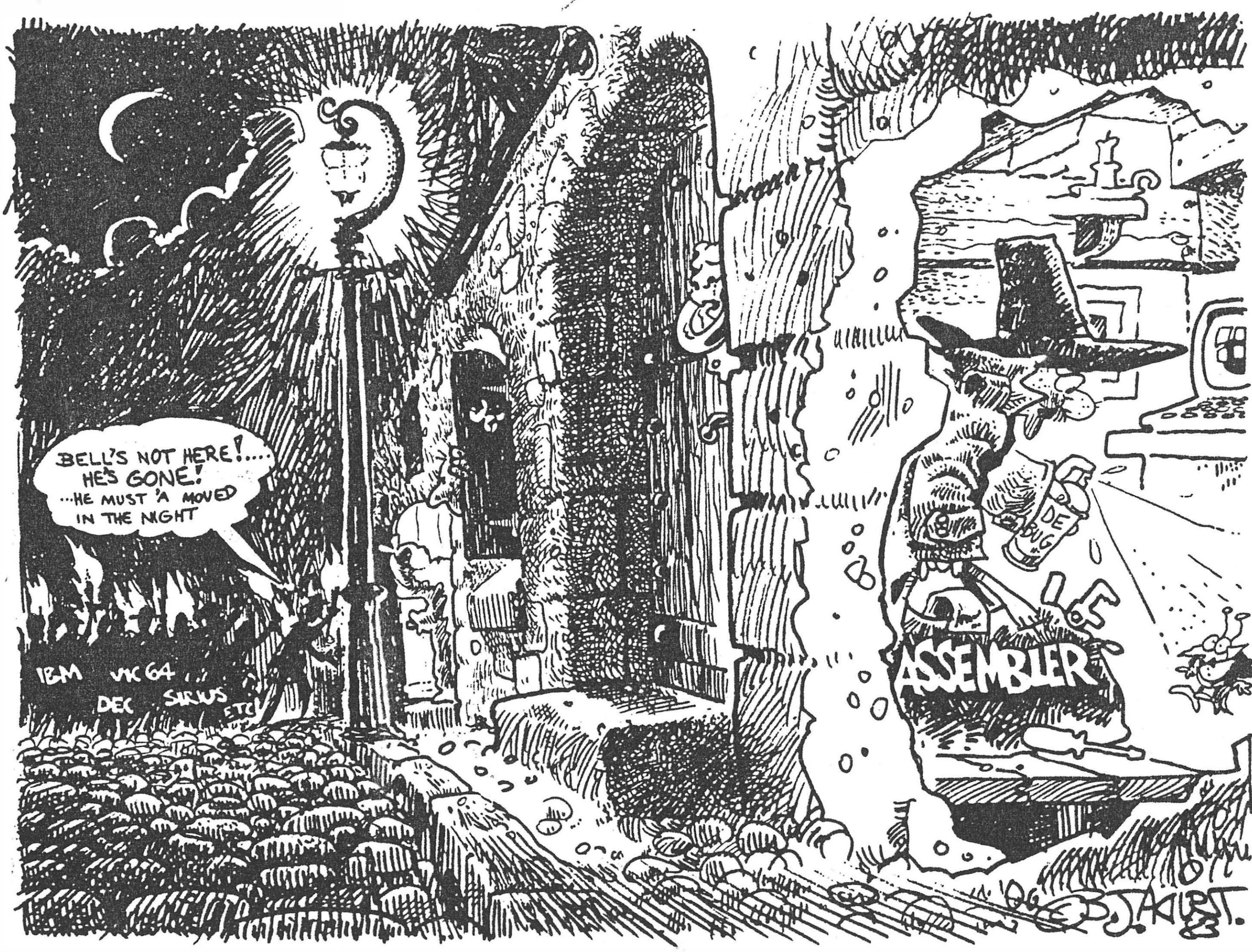
Next month, breakpoints. □

p 62.

2 of 2. YC Mar 83.

your computer tutorial

Understanding Assembler Part XII



Fresh from choosing the Personal Computer of the Year, then moving office (to escape the howling computer companies?), Les Bell takes up where he left off in our series...

WELL, DEAR READER, when last we spoke, we were discussing monitor programs, and had designed one with the rudimentary functions of dumping memory, changing memory and running programs. However, it wasn't much use for really debugging programs, as it lacked the ability to set and remove breakpoints.

Breakpoints are points in a program under development where you want to break out of the program and examine the state of memory and the processor registers. In assembly language, this is most easily done by replacing an instruction with a jump to the monitor, which then saves the processor registers and dumps them to the screen.

Because the shortest instruction in the 8080 set is only one byte long, we'll have to replace it with a one-byte jump instruction. Fortunately, such an instruction does exist: the **RSTn** instruction, which allows us to call one of eight locations in memory with only a one-byte instruction.

What our debugger must do, then, is store away the original instruction from the chosen breakpoint location, and replace

1 of 6.

YC Aug 83.

it with an RST instruction – in this case, RST 5. (If you're wondering why I chose RST 5, it's because SID already uses RST 7, and MP/M uses RST 6, so I couldn't use those – otherwise how would I debug the debugger?)

Here's the breakpoint setting routine itself. Nothing very tricky here:

```
*****************************************************************
*                                                               *
*                     breakpoint command                        *
*                                                               *
*****************************************************************

; set a breakpoint at an address given in command line

break:  call    spskip          ; skip over spaces
        call    getparm         ; get address of breakpoint
        mov     a,m             ; get the original instruction
        sta     instr           ; save it away
        mvi     m,0efh          ; replace it with a restart 5
        shld    tempad          ; save the breakpoint address
        ret
```

The RST 5 instruction will cause the program to, in effect, CALL location 0026H, and so we must ensure that our cold-start code places a JMP there to the debugger code, which saves the processor registers and prints them. In this case, I've called this routine TRAP. This little piece of code does that job:

```
mvi     a,(jmp)         ; jump for rst 5
sta     0028h
lxi     h,trap
shld    0029h
```

Now for the *piece de resistance:* the trap routine itself:

```
*****************************************************************
*                                                               *
*                        trap routine                           *
*                                                               *
*****************************************************************

; encountering a breakpoint sends the processor here to
; print the contents of the registers
```

The first thing the routine does is to exchange HL with the top of the stack. This does two things: first, it saves HL on the stack; secondly, it fetches the return address of the RST 5 instruction into HL (remember, RST 5 is really a CALL which places the return address on the stack).

Now, the return address is the address *after* the breakpoint, so we must decrement the program counter before storing it away. Next, we push the remaining registers on the stack, so that they are safely beyond harm's reach on the stack:

```
trap:
xthl                    ; get breakpoint address
dcx     h               ; pc is one too high
shld    tempad          ; save the breakpoint address
push    d
push    b
push    psw
```

Having done this, we can load HL with the value of the stack pointer and use it to read the register values off the stack before printing them:

```
lxi     h,0
dad     sp
```

Then we get the first byte off the stack. This byte is half the program status word, and contains the flags:

```
mov     a,m             ; A contains flags
call    flprt           ; print them
```

The easiest way to display the flags is with a general purpose routine which displays any byte in binary:

```
flprt:  mvi     d,8             ; set counter to number of bits
fl1:    ral
        mov     b,a             ; save flags in B
        mvi     a,'0'           ; get an ASCII zero
        aci     0               ; and if carry is set, make '1'
        mov     c,a             ; then put it in C
        call    putch
        mov     a,b             ; retrieve flags
        dcr     d               ; count down
        jnz     fl1             ; and loop again
        ret
```

The only real trick in this routine is the use of the carry bit to decide whether to print 1 or 0; I've never seen this technique used elsewhere, but I'm sure someone else must have thought of it.

The next task is to print the contents of the accumulator, which is the next byte on the stack. It might be a good idea, from this point on, to label what we print across the screen, so I included short messages in the code and wrote a simple in-line print routine to handle them:

```
ilprt:                          ; in-line print subroutine
        xthl                    ; get ptr and save in HL
ill:    mov     a,m             ; get char
        ora     a               ; reached end?
        jz      ilex            ; yes, exit
        mov     c,a             ; move into C
        call    putch           ; and print
        inx     h               ; point to next
        jmp     ill             ; and go round
ilex:   inx     h               ; point to byte after end 0
        xthl                    ; restore HL
        ret                     ; and return
```

Notice that when this routine is entered, the top of the stack is the return address, which (not entirely by accident) is also the address of the first byte of the message to be printed. This byte is checked to see if it's zero (the string terminator), then output; and so we continue through the string. Finally, HL points beyond the string, and we stick it back on the stack and do a return – to the first instruction after the string.

This technique, therefore, requires the message to be written in the code, immediately after the "call ilprt" instruction:

```
call    ilprt           ; print message
db      ' A= ',0
inx     h               ; point to 'A' on stack
mov     a,m             ; get the value
call    h8              ; and print it
```

And so it continues, now with the remaining register pairs. Notice that the 8080 places 16-bit values in memory with the two bytes in reverse order. So I wrote a dead-simple routine to get the two bytes in reverse order and print them:

```
call    ilprt
db      ' BC= ',0       ; print BC
call    trl             ; print register pair from memory
call    ilprt
db      ' DE= ',0       ; ditto DE
call    trl
call    ilprt
db      ' HL= ',0       ; ditto HL
call    trl
```

Finally, we know that we have saved four register pairs since encountering the RST 5 instruction. Therefore, by adding to the current value of the stack pointer, we should (and do) have the value of SP just before the breakpoint:

```
call    ilprt
db      ' SP= ',0
lxi     h,8             ; number of registers saved
dad     sp              ; that's original stack value
call    h16             ; print it
```

We also know the value of the breakpoint, as it was stored in tempad as we entered the breakpoint routine. So we print it, and also save the address ready to resume – but it must be saved in a new address in case we set another breakpoint.

2 of 6 YC Aug 83

Finally we restore the instruction which the breakpoint had displaced, and jump to the main monitor routine:

```
        call    ilprt
        db      ' PC= ',0
        lhld    tempad          ;
        call    h16
        lhld    tempad          ; get breakpoint address
        shld    lastbrk         ; save for resume
        lda     instr           ; get instruction
        mov     m,a             ; and restore it
        jmp     monl            ; go to mainline

trl:    inx     h               ; increment past E 'cos
        inx     h               ; DE reversed on stack
        mov     a,m             ; get D
        call    h8
        dcx     h               ; point to E
        mov     a,m             ; get it
        call    h8
        inx     h               ; increment past D
        ret
```

The resume command allows us to continue execution after a breakpoint. It simply restores the registers by popping them in reverse order. Bear in mind that the scanner called this routine, so there is an extra return address on the stack which we must get rid of first:

```
*********************************************************************
*                                                                   *
*                          resume command                           *
*                                                                   *
*********************************************************************

resume:                         ; resume operation after break
        pop     psw             ; pop extra return address
        pop     psw
        pop     b
        pop     d
        lhld    lastbrk         ; get breakpoint address
        xthl                    ; get return address
        ret                     ; and go there
```

The only thing left to do is to allocate some storage for the various variables we have used; this should be placed at the end of the program:

```
instr   ds      1
tempad  ds      2
lastbrk ds      2
```

There are several improvements that could be made to this program. The first glaring omission is that the debugger makes no attempt to maintain a separate stack for its own use; it simply sticks registers and its own internal return addresses on to the stack of the program it is debugging. While this is all right if that program is using the debugger's stack, which allows 32 levels of pushing, you should be aware whether or not your programs maintain their own stack.

It is possible to rewrite the trap and resume routines to switch stacks, and this is left as an exercise for the reader (my way of saying, "Why should I do all the work?").

The next major improvement would be to add single-stepping, which is done by continually inserting breakpoints. The major difficulty here is that the 8080 instructions vary in length - one, two or three bytes – and the debugger must know how long each is, in order to place a breakpoint after it. Nonetheless, it can be done (how do you think DDT works?).

Meanwhile, bear in mind that it's possible to write your program with multiple RST 5's already in the code for debugging purposes. Once the program is debugged, you can take out those restarts.

Another improvement would be to tidy up the flag-printing routine to label each flag, or do what SID does: print dashes for reset flags, and initials for the ones that are set.

What happens if you resume before encountering a breakpoint? Perhaps it might be a good idea to prevent that, and while you're at it, extend the idea so that you can only resume once after a breakpoint.

That really wraps up all the elementary features of a monitor program. In designing this program, we have used a number of techniques and programming tricks. The code has been fairly modular, so the program contains a number of subroutines which may be helpful to you in your own programming.

In the next article, I'll move on to start on file input/output under CP/M, using the construction of a word-counting program as an example. Meanwhile, here's the completed monitor program:

```
; 8080 / Z-80 monitor routines
;

false   equ     0
true    equ     not false

rmac    equ     false

ctrc    equ     03h                     ; control-C for abort
acr     equ     0dh                     ; carriage return
alf     equ     0ah                     ; line feed
ctrh    equ     08h                     ; backspace character
tab     equ     09h
ctrs    equ     13h                     ; control-S for pause output
ctrx    equ     18h                     ; control-X erases input line
clear   equ     1ah                     ; TVI 910 clear screen cmd

del     equ     7fh

prompt  equ     '>'
buflen  equ     128
exit    equ     0000h

        title   'monitor rev 1.4'

        if      not rmac
        org     0100h
        endif

monitor:
        jmp     boot
monl:   call    crlf
        mvi     c,prompt
        call    putch
        lxi     h,buff
        call    getln
        lda     buff
        cpi     acr
        cnz     scanner
        jmp     monl

getln:  mvi     e,0             ; character counter
getlnl: call    conin           ; get a character
        cpi     ' '             ; is it a control char?
        jc      getln4          ; yes, jump to handler
        cpi     'Z'+1           ; is it lower case?
        jc      getln2          ; no, carry on regardless
        ani     5fh             ; otherwise fix it
getln2: mov     m,a             ; store char in buffer
        mvi     a,buflen        ; get buffer length
        cmp     e               ; have we reached it?
        rz                      ; yes, return to caller
        mov     a,m             ; retrieve character
        inx     h               ; bump buffer pointer
        inr     e               ; and counter
        mov     c,a
        call    putch           ; echo char
        jmp     getlnl

getln4: cpi     ctrh
        jz      getln5
        cpi     ctrx
        jz      clline
        cpi     acr
        jnz     getlnl
        mov     m,a

crlf:   mvi     c,acr
        call    putch
        mvi     c,alf
        call    putch
        ret
```


3 of 6   YC   Aug 83.

```
getln5:                         ; control-H (backspace) handler
        mov     a,e
        ora     a
        jz      getln1
        mvi     c,ctrh          ; print backspace
        call    conout
        mvi     c,' '           ; then a space
        call    conout
        mvi     c,ctrh          ; then another backspace
        call    conout
        dcr     e               ; count down
        dcx     h               ; back up buffer pointer
        jmp     getln1          ; get next character

clline:                         ; control-X (clear line) handler
        mvi     c,ctrh          ; print backspace
clline1:
        mov     a,e
        ora     a
        jz      getln
        call    conout
        mvi     c,' '           ; print space
        call    conout
        mvi     c,ctrh          ; print backspace
        call    conout
        dcx     h               ; back up buffer pointer,
        dcr     e               ; count back a char
        jnz     clline1         ; to beginning of line
        jmp     getln           ; and start all over
putch:
        call    conout
        call    const           ; test for input character
        ora     a               ; set flags to test for zero
        rz                      ; no character, so return
        call    conin
        cpi     ctrs            ; control-S?
        jz      pawz            ; yes, pause
        jmp     mon1            ; no, halt
pawz:   call    conin
        cpi     ctrc            ; control-C?
        jz      mon1            ; yes, abort
        ret

        page
*****************************************************************
*                                                               *
*                       simple scanner                          *
*                                                               *
*****************************************************************

scanner:
        lxi     h,buff
        mov     a,m             ; get a character
        inx     h               ; point to next
        shld    cursor          ; and save cursor
        sui     'A'             ; convert to table index
        jc      error           ; if less than A then error
        cpi     'Z' - 'A' + 1   ; if > 'Z' then error
        jnc     error
        add     a               ; double A
        lxi     h,table         ; point to start of table
        mvi     d,0
        mov     e,a             ; put offset in DE
        dad     d               ; and add it to HL
        mov     e,m             ; get low byte
        inx     h
        mov     d,m             ; get high byte
        xchg                    ; get it into HL
        pchl

table:
        dw      error           ; A
        dw      break           ; B
        dw      error           ; C
        dw      dump            ; D
        dw      exit            ; E
        dw      error           ; F
        dw      go              ; G
        dw      error           ; H
        dw      error           ; I
        dw      error           ; J
        dw      error           ; K
        dw      error           ; L
        dw      error           ; M
        dw      error           ; N
        dw      error           ; O

        dw      error           ; P
        dw      error           ; Q
        dw      resume          ; R
        dw      subst           ; S
        dw      error           ; T
        dw      error           ; U
        dw      error           ; V
        dw      error           ; W
        dw      error           ; X
        dw      error           ; Y
        dw      error           ; Z

*****************************************************************
*                                                               *
*                       error handler                           *
*                                                               *
*****************************************************************

error:
        mvi     c,'?'
        call    conout
        call    crlf
        jmp     wboot

        page
*****************************************************************
*                                                               *
*                       scanner tools                           *
*                                                               *
*****************************************************************

spskip:                         ; skip over spaces and delimiters
        lhld    cursor
spsk1:  mov     a,m
        inx     h
        cpi     acr             ; no cr expected, error
        jz      error
        cpi     ' '
        jz      spsk1
        cpi     ','
        jz      spsk1
        cpi     tab
        jz      spsk1
        dcx     h
        shld    cursor
        ret

getparm:                        ; get hex parameter
        push    d
        push    b
        lxi     h,0
gp1:    call    getchar
        call    gp2
        jc      gp3
        dad     h
        dad     h
        dad     h
        dad     h
        ora     l
        mov     l,a
        jmp     gp1

gp2:

; convert ASCII to binary, return with carry set if not a valid digit.

        sui     '0'             ; remove ASCII offset
        rc                      ;  character < 0
        cpi     'F'-'0'+1
        cmc                     ; complement carry
        rc                      ; character > F
        cpi     10
        cmc                     ; complement carry
        rnc                     ; ok, number 0 - 9
        sui     'A'-'9'-1
        ret

gp3:                            ; character not hex
        call    ungetch
        pop     b
        pop     d
        ret

getchar:                        ; get character from buffer
        push    h               ; save HL
        lhld    cursor          ; get cursor
        mov     a,m             ; get character
        inx     h               ; point to next
        shld    cursor          ; save cursor
        pop     h               ; restore HL
        ret

ungetch:                        ; back up cursor
        push    h               ; save HL
        lhld    cursor          ; get cursor
```

4 of 6   YC   Aug 83.

```
        dcx     h               ; move back
        shld    cursor          ; save cursor
        pop     h               ; restore HL
        ret

        page

;*****************************************************************
;*                                                               *
;*                            dump                               *
;*                                                               *
;*****************************************************************

dump:
        call    spskip
        call    getparm         ; get start address
        xchg                    ; put start into DE
        call    spskip
        call    getparm         ; get finish
        xchg                    ; DE <- finish, HL <- start
d0:     push    h               ; save base pointer on stack
        call    hl6             ; print initial address
        mvi     c,tab           ; and tab
        call    putch
d1:     mov     a,m             ; get byte from memory
        call    h8
        mvi     c,' '           ; print a space
        call    putch
        inx     h               ; point to next byte
        call    d8              ; have we reached the end?
        jc      d2              ; dump remaining ascii
        mov     a,l
        ani     0fh             ; mask lower bits
        jnz     d1              ; if not zero, keep dumping
d2:     mvi     c,' '           ; else space and keep dumping
        call    putch
        pop     h               ; get base pointer
d4:     mov     a,m             ; get char from memory
        ani     7fh             ; strip msb
        cpi     ' '             ; if less than space
        cm      d7              ; replace with a dot
        mov     c,a             ; move into C
        call    putch           ; and print it
        inx     h               ; point to next
        call    d8
        jc      crlf            ; exit with a CRLF
d5:     mov     a,l
        ani     0fh
        jnz     d4
        call    crlf
        jmp     d0

d7:     mvi     a,'.'
        ret

d8:                             ; reached end yet?
        mov     a,e
        sub     l
        mov     a,d
        sbb     h
        ret

        page

;*****************************************************************
;*                                                               *
;*                   S (substitute) command                      *
;*                                                               *
;*****************************************************************

; display memory with option of changing, byte by byte

subst:
        call    spskip          ; skip over spaces
        call    getparm         ; get start address
subst1:
        call    hl6             ; output address
        mvi     c,' '           ; and then a space
        call    putch
        push    h               ; save address on stack
        mov     a,m             ; get memory contents
        call    h8              ; output byte in A
        mvi     c,' '           ; space in C
        call    putch           ; and output it
        lxi     h,buff          ; point to buffer
        call    getln           ; get a hex number (or whatever)
        lxi     h,buff          ; point to beginning of buffer
        shld    cursor          ; set cursor there
        mov     a,m             ; get first char
        cpi     '.'             ; is it a dot?
        jz      mon1            ; back to mainline
        cpi     acr             ; if it's a cr, don't change
        jz      subst2          ; skip over change code
        call    getparm         ; get hex number
        mov     a,l             ; mov byte into a
        pop     h               ; get back address
        mov     m,a             ; and store byte
        inx     h               ; move on to next
        jmp     subst1          ; and loop round
subst2:
        pop     h               ; restore stack
        inx     h               ; move on
        jmp     subst1          ; and loop round

        page

;*****************************************************************
;*                                                               *
;*                        go command                             *
;*                                                               *
;*****************************************************************

; go (jump) to an address from command line

go:     call    spskip          ; skip over spaces
        call    getparm         ; get address to go to
        xchg                    ; save in DE
        lxi     h,wboot         ; get return address
        push    h               ; and place it on stack
        xchg                    ; get go address again
        pchl                    ; and go

;*****************************************************************
;*                                                               *
;*                    breakpoint command                         *
;*                                                               *
;*****************************************************************

; set a breakpoint at an address given in command line

break:  call    spskip          ; skip over spaces
        call    getparm         ; get address of breakpoint
        mov     a,m             ; get the original instruction
        sta     instr           ; save it away
        mvi     m,0efh          ; replace it with a restart 5
        shld    tempad          ; save the breakpoint address
        ret

        page

;*****************************************************************
;*                                                               *
;*                      trap routine                             *
;*                                                               *
;*****************************************************************

; encountering a breakpoint sends the processor here to
; print the contents of the registers

trap:
        xthl                    ; get breakpoint address
        dcx     h               ; pc is one too high
        shld    tempad          ; save the breakpoint address
        push    d
        push    b
        push    psw
        lxi     h,0
        dad     sp
        mov     a,m             ; A contains flags
        call    flprt           ; print them
        call    ilprt           ; print message
        db      ' A= ',0
        inx     h               ; point to 'A' on stack
        mov     a,m             ; get the value
        call    h8              ; and print it
        call    ilprt
        db      ' BC= ',0       ; print BC
        call    trl             ; print register pair from memory
        call    ilprt
        db      ' DE= ',0       ; ditto DE
        call    trl
        call    ilprt
        db      ' HL= ',0       ; ditto HL
        call    trl
        call    ilprt
        db      ' SP= ',0
        lxi     h,8             ; number of registers saved
        dad     sp              ; that's original stack value
        call    hl6             ; print it
        call    ilprt

        db      ' PC= ',0
```


5 of 6 YC Aug 83

```
        lhld    tempad
        call    h16
        lhld    tempad          ; get breakpoint address
        shld    lastbrk         ; save for resume
        lda     instr           ; get instruction
        mov     m,a             ; and restore it
        jmp     monl            ; go to mainline

trl:    inx     h               ; increment past E 'cos
        inx     h               ; DE reversed on stack
        mov     a,m             ; get D
        call    h8
        dcx     h               ; point to E
        mov     a,m             ; get it
        call    h8
        inx     h               ; increment past D
        ret

flprt:  mvi     d,8             ; set counter to number of bits
fll:    ral
        mov     b,a             ; save flags in B
        mvi     a,'0'           ; get an ASCII zero
        aci     0               ; and if carry is set, make '1'
        mov     c,a             ; then put it in C
        call    putch
        mov     a,b             ; retrieve flags
        dcr     d               ; count down
        jnz     fll             ; and loop again
        ret

ilprt:                          ; in-line print subroutine
        xthl                    ; get ptr and save in HL
ill:    mov     a,m             ; get char
        ora     a               ; reached end?
        jz      ilex            ; yes, exit
        mov     c,a             ; move into C
        call    putch           ; and print
        inx     h               ; point to next
        jmp     ill             ; and go round
ilex:   inx     h               ; point to byte after end 0
        xthl                    ; restore HL
        ret                     ; and return

*********************************************************************
*                                                                   *
*                       resume command                              *
*                                                                   *
*********************************************************************

resume:                         ; resume operation after break
        pop     psw             ; pop extra return address
        pop     psw
        pop     b
        pop     d
        lhld    lastbrk         ; get breakpoint address
        xthl                    ; get return address
        ret                     ; and go there

        page

*********************************************************************
*                                                                   *
*                     hex output routines                           *
*                                                                   *
*********************************************************************

h16:
        mov     a,h
        call    h8
        mov     a,l
h8:
        push    psw
        rrc
        rrc
        rrc
        rrc
        call    h4
        pop     psw
h4:
        ani     0fh
        adi     '0'
        cpi     '9'+1
        cp      hadj
        mov     c,a
        call    putch
        ret
hadj:   adi     'A'-1-'9'
        ret

        page
```


6 of 6 YC Aug 83.

```
*********************************************************************
*                                                                   *
*                          fake bios                                *
*                                                                   *
*********************************************************************

data    equ     0e3f8h          ;data port
stat    equ     0e3f9h          ;status port
rbf     equ     00000100b       ;receive buffer full bit
tbe     equ     00001000b       ;transmit buffer empty bit

pio     equ     20h             ; parallel I/O chip (8255A)
lstat   equ     pio+2
ldata   equ     pio+1
pcont   equ     pio+3

; JUMP TABLE
fbios:  jmp     boot
        jmp     wboot
        jmp     const
        jmp     conin
        jmp     conout
        jmp     list
        jmp     punch
        jmp     reader

boot:   mvi     a,10101100b     ; set 8255 mode
        out     pcont
        mvi     a,00000101b     ; set group b mode
        out     lstat
        mvi     a,(jmp)         ; jump for rst 5
        sta     0028h
        lxi     h,trap
        shld    0029h
        mvi     c,clear
        call    conout

wboot:  lxi     sp,stk
        jmp     monl

; console status routine, returns 0 if no char, 0FFH if character avail

const:  lda     stat
        ani     rbf             ; mask rbf bit
        mvi     a,0             ; if no data, return with zero in A
        rnz
        dcr     a               ; otherwise put 0ffh in A
        ret                     ; and return

; console input routine

conin:  lda     stat            ; get status from UART
        ani     rbf             ; mask rbf bit
        jnz     conin           ; wait for character
        lda     data            ; get character
        cma                     ; adjust for negative bus
        ani     7fh             ; strip high bit
        ret

; console output routine

conout: lda     stat            ; get status from UART
        ani     tbe             ; mask tbe bit
        jnz     conout          ; wait for buffer to empty
        mov     a,c             ; move character into A
        cma                     ; adjust for negative bus
        sta     data            ; and send it
        cma                     ; make positive again
        ret

list:
        in      lstat           ; get lp status
        ani     2
        jz      list
        mov     a,c
        out     ldata
        ret

; other functions dummies for this example

punch:
reader:
        ret                     ; return

buff    ds      128
        ds      64
stk     equ     $
cursor  ds      2
instr   ds      1
tempad  ds      2
lastbrk ds      2
        end
```

your computer tutorial

Understanding Assembler
Part XIII

We've heard a lot about word counters in recent issues, so with an eye to the main chance, Les Bell encourages his followers to try out that kind of program in assembler in the latest chapter of his exciting series ...

THE LAST COMMAND I used on my computer before starting to write this article was a word-counting program. I'd just finished writing an article, checked the spelling and wanted to check whether the article was about the right length (it was).

Word counting is a task the professional writer has to perform from time to time. In general, counting pages is good enough, but just occasionally you have to know exactly how long a piece is. Having the computer do the hack work makes life easier all round; where before you would never bother checking the length of many stories, now you can check them all.

It also happens that a word counting program would make an interesting project in assembly language, for several reasons. First, I've already written such a program in a couple of high-level languages, so I've got the overall design down pat. This is important; often it's easier to design a program in a high-level language first, and then re-write it in assembler.

This dovetails neatly with the concept of structured program design. The thinking here is that the program should be designed using a pseudo-language, in fairly vague terms at first, but with successive refinements until one has a detailed model on which to base the program proper.

Thus, for a word counting program, our program design might start out as:

```
count words
```

which is not really very helpful, but at least we've written down some kind of objective to get started. A journey of a thousand miles begins with one step.

A first refinement might be:

```
initialise counter
open input file
do while not end-of-file
get a word
add 1 to the counter
end-while
print value of counter
```

Now we're getting somewhere. We've split the program up into a number of elements, some of which we have probably written before. The whole project is a little less daunting. Successive refinements will deal with each line of this version. In particular, we must deal with error conditions: what do we do if the input file does not exist? What do we do if the user does not specify an input file?

We must also figure out how we are going to get a word from the input file. In assembly language, we don't have the facilities provided by higher-level languages for breaking up input. In fact, just processing the input will turn out to be one of the trickiest areas of this program.

The CP/M operating system, which we will use, reads data from a disk file in 128-byte records. So when we 'get' from the file, we don't get a word, we get 128 characters. Furthermore, the 8080/Z-80 microprocessor will only deal with one character at a time in the accumulator, so at this level we're dealing with characters, and not words.

The answer in this case is to examine the file character by character using some rule to decide when to count a word. The answer lies not in figuring out what constitutes a word, but in what doesn't make a word. In ASCII text, words are separated by only a few distinct characters: spaces, tabs, carriage returns and line feeds. You could optionally include hyphens, to count hyphenated words as two; all other punctuation – for our purposes – simply makes words longer.

Thus, while we are reading a file, we start off outside a word, and as we read, as long as we see any of those four characters, we are still outside a word. If we see anything else, we make the transition from outside a word to inside, and it is these transitions that we count. When next we see a space, or other non-word character, we are back outside the word again.

I first saw this technique in Kernighan and Ritchie's excellent book, *The C Programming Language*, and when I got the BDS C compiler, it was the first really useful program I got up and running. BDS C has somewhat non-standard file input/output, and so it was an exercise in mastering those features of the language.

Advantages of 'C'

The advantage of writing the program in C is that you have all the benefits of structured design using a pseudo-language, but the program can in fact be compiled and run in order to test the logic of the design. Now, sometimes you will find that the high-level language version of the program is adequate, in which case there is no need to rewrite it in assembler.

However, other times you will find the program is too slow, but that there is an innermost loop, doing most of the work, which can be rewritten in assembler and linked to the remainder in the high-level language. This is the approach I took with the fog index calculator (another writer's tool) in *YC* December 1982.

If the worst comes to the worst, a complete rewrite is necessary – but at least you have completely proved the logic of your design and can code directly from the HLL version. This leads naturally to the best way to write assembler – as little as possible!

For those who want to relate the assembler design to the C version, here is the original. For those who find C rather cryptic (most of us) I've tried to annotate it extensively. Everything in /*...*/ is a comment.

Hopefully it's not too cryptic, and you should be able to see a relationship to the early pseudo-code design. The ▶

```c
#include        <bdscio.h>

#define YES     1
#define NO      0
#define EOF     0x1a    /* CP/M end of file character */
#define ERROR   -1
#define MASK    0x7f    /* Used to strip out high bit of
                           WordStar files */

main(argc,argv) /* count lines, words, chars in input file */
char    **argv;
{
        int c, inword, fd;     /* current character, word flag,
                                  file descriptor */
        unsigned nl, nw, nc;   /* number of lines, words, chars */
        char    buf[BUFSIZ];   /* file buffer */

        if(argc != 2) {        /* User probably just typed 'WC' */
                printf("Usage: wc filename\n");
                exit();
        }

                               /* If file doesn't exist */
        if((fd = fopen(argv[1],buf)) == ERROR) {
                printf("cannot open: %s\n",argv[1]);
                exit();
        }
                               /* Initialise counters */
        inword = NO;
        nl = nw = nc = 0;
         /* Get a char, check for end of file, strip high bit */
        while ((c = (getc(buf) & MASK)) != EOF)
        {
                ++nc;          /* count a char */
                if (c == '\n') /* if a line feed, count a line */
                        ++nl;
                               /* If space, tab, CR or LF */
                if (c == ' ' || c == '\n' || c == '\t' || c == 0x0d)
                        inword = NO;
                else if (inword == NO) {
                        inword = YES;
                        ++nw; /* count a word */
                }
        }

                               /* print results */
        printf("number of lines = %u\n",nl);
        printf("number of words = %u\n",nw);
        printf("number of chars = %u\n",nc);
}
```

Original listing in 'C'.

BDOS Function Summary

Func	Function Name	Input Parameters	Output Results
0	System Reset	none	none
1	Console Input	none	A = char
2	Console Output	E = char	none
3	Reader Input	none	A = char
4	Punch Output	E = char	none
5	List Output	E = char	none
6	Direct Console I/O	see def	see def
7	Get I/O Byte	none	A = iobyte
8	Set I/O Byte	E = iobyte	none
9	Print String	DE = &buffer	none
10	Read Console Buffer	DE = &buffer	see def
11	Get Console Status	none	A = 00/ff
12	Return Version Number	none	A = version
13	Reset Disk System	none	see def
14	Select Disk	E = disk number	see def
15	Open File	DE = &fcb	A = dir code
16	Close File	DE = &fcb	A = dir code
17	Search for First	DE = &fcb	A = dir code
18	Search for Next	DE = &fcb	A = dir code
19	Delete File	DE = &fcb	A = dir code
20	Read Sequential	DE = &fcb	A = err code
21	Write Sequential	DE = &fcb	A = err code
22	Make File	DE = &fcb	A = dir code
23	Rename file	DE = &fcb	A = dir code
24	Return Login Vector	none	HL = login vect
25	Return Current Disk	none	A = disk no
26	Set DMA Address	DE = &dma	none
27	Get &alloc_vec	none	HL = &alloc_vec
28	Write Protect Disk	none	see def
29	Get R/O Vector	none	HL = R/O vector
30	Set File Attributes	DE = &fcb	see def
31	Get &dpb	none	HL = &dpb
32	Set/Get User Code	see def	see def
33	Read Random	DE = &fcb	A = err code
34	Write Random	DE = &fcb	A = err code
35	Compute File Size	DE = &fcb	fcb.ranrec, fcb.ovflow
36	Set Random Record	DE = &fcb	fcb.ranrec, fcb.ovflow

Table 1. BDOS functions. Notes: 'see def' means that the reader is referred to the Digital Research manual CP/M 2.0 Interface Guide for full details. The '&' symbol means 'address of'; 'fcb' = 'file control block'; 'dpb' = 'disk parameter block'; 'allocvec' = 'allocation vector'; 'fcb.ranrec' and 'fcb.ovflow' refer to the random record number of a file control block.

job now is to translate this into assembler. Fortunately it's not too difficult, except for those areas where the C compiler or function library does something for us automatically, like file buffering or printing a decimal number.

CP/M File Access

So far in this series, we haven't had to interface anything to CP/M, so before we start coding up this example it seems appropriate to provide a short tutorial on the CP/M file system.

CP/M is split up into three major parts: the BIOS (Basic Input/Output System), which is the hardware-dependent part; the BDOS (Basic Disk Operating System), which is the logical part of the disk operating system proper; and the CCP (Console Command Processor) which is the program that puts up the A› prompt, contains the TYPE, ERA and other commands, and will load and run your programs.

The BIOS does not concern us here. We are primarily concerned with the BDOS functions for opening and reading files and writing to the console, as well as with a couple of services the CCP performs for us.

The BDOS is entered via a single jump, located at 0005H, down at the bottom of memory. The user tells the BDOS which function he/she wants performed by passing the function number in the C register, and where necessary passing any data in the DE register pair. If the BDOS returns a result it will be in the HL register pair, in the case of a 16-bit value or address, or in the accumulator for characters, error codes or other single-byte values.

What are these functions? Table 1 lists the BDOS functions, together with values passed and returned. The '&' symbol, by the way, indicates 'address of', as in the C programming language.

The functions of most interest to us are numbers 2 (console write), 9 (print string), 15 (open file) and 20 (read next record). The program will use these functions to print results and to read the text input from a disk file.

The operation of the CCP is also very important, as it will do some useful work for us. When you type 'WS *filename*' at the CCP command line in order to edit a file, the CCP does a number of things.

First it locates WS.COM on the current disk and loads it. That much is obvious. It also examines the filename typed after 'WS', and translates it into the standard CP/M form, with eight characters before the point and three after, with spaces padding out empty character positions. It then places this filename into a special area of memory called a file control block, which will be used by CP/M to keep track of the file. The FCB is always located at address 005CH (there's another at 006CH which is also initialised by the CCP if necessary).

The CCP will also set up the BDOS to transfer from files into a default buffer area at location 0080H, which is the 128 bytes just below where our program resides, at 0100H. This is the buffer we shall use for this program; it saves us having to tell the BDOS we shall be using another.

Finally, the CCP copies the command line tail (the filename in this case) into the first few bytes of that buffer, so that we can examine it to see if any options have been specified by the user or whatever. All this is done by the CCP before it hands control over to our program.

Next month, the program itself. □

2 of 2. YC Oct 83.

your computer tutorial

Understanding Assembler – Part XIV

Continuing last month's exercise of designing a word counting program, Les Bell discusses CP/M file handling, structured design and recursive programming.

LAST MONTH we looked at the conceptual design of a word counting program, and showed a possible design in the C programming language. This month we'll continue with the actual assembly language program.

Having written the program and tested it in C first, the actual writing of the assembler version is not difficult at all. In fact, I did it by copying the C version source file, renaming it to .ASM, and editing it heavily. The knack is to think like a compiler, and don't get tricked into trying to save a few bytes of code by leaving variables in processor registers and pushing and popping them or any other tricky coding practices.

Instead, just set up variables in memory and load and store them directly; that's exactly what most high-level languages do, except those on 16-bit processors which are able to use stack frames – not easy on the 8080 or Z80.

Without further ado, let's examine the program. It starts off in a quite straightforward manner. First we define a few logical and other constants:

```
yes     equ     1               ; logical values
no      equ     0
false   equ     0
true    equ     not false
tab     equ     09h             ; ASCII characters
acr     equ     0dh
alf     equ     0ah
mask    equ     7fh             ; to take out high bit of WS docs
```

Then we define various CP/M addresses, function numbers and returned values:

```
defdma  equ     0080h           ; CP/M default DMA address
fcbl    equ     05ch            ; CP/M file control block
ex      equ     12              ; extent number
s2      equ     14              ; who knows?
cr      equ     32              ; current record
eof     equ     1ah             ; CP/M end of file marker
bdos    equ     0005h           ; bdos entry point
error   equ     -1              ; BDOS return error code
warm    equ     0               ; CP/M warm start entry
conout  equ     2               ; console output function
openf   equ     15              ; CP/M BDOS open file function
readrec equ     20              ; CP/M BDOS read record function
setdmaf equ     26              ; set DMA address
```

Then comes the start of the program. The first block of code saves the stack pointer value which was set up by the CCP (CP/M's Console Command Processor), since this program will make extensive use of the stack at one point, and we don't want the stack pointer to overrun vital parts of the BDOS.

```
        org     0100h

start:  lxi     h,0             ; set up local stack
        dad     sp
        shld    oldsp           ; save old stack
        lxi     sp,stk
```

The next section takes care of the possible error conditions on the command line. First we check that there is a file name in the command line. Since the CCP will copy the command line tail (everything after the command and the space that follows it) into the first few bytes of the default disk buffer at 0080H, all we have to do is check that the first character (at 0080H) is not a space:

```
wc:     lda     defdma          ; examine first character of CP/M
        cpi     ' '             ; command tail to check for non-
        jnz     wc1             ; existent file name
        call    ilprt
        db      'Usage: wc filename',acr,alf,0
        jmp     warm
```

Next we try to open the input file by calling a subroutine which will do this for us. The subroutine returns a value in A; if this is 255 (error) then something is wrong – usually the file is not on the specified (or more usually default) disk drive.

```
wc1:
        call    fopen           ; try to open file
        cpi     error
        jnz     wc2
        call    ilprt
        db      'Cannot open file',acr,alf,0
        jmp     warm
```

Now comes the program proper. We start off by setting 'inword' to NO, then get the first character by calling a subroutine. We then strip off the most significant bit, since it should not be set in the ASCII code, but WordStar and other word processors are apt to use it for their own purposes. Then we check that it is not a control-Z, which is the CP/M end of file character. If it is, then we jump to the part of the program which prints the results.

```
wc2:
        call    inno            ; inword = NO
wc3:    call    getc            ; get a character
        ani     mask            ; strip high bit
        cpi     eof             ; is it end of file?
        jz      wc9             ; if so, print results
```

Assuming that we have a valid character in the accumulator, we then proceed to count it, which is simply a matter of loading the current count into HL and performing an increment instruction, then restoring the value.

```
        lhld    nc              ; nc = nc + 1
        inx     h
        shld    nc
```


1 of 3.

YC Nov 83.

Next we must check whether the character marks an end of line. If it does, we then increment the number of lines by calling a subroutine, which works in just the same way as the code above. Under CP/M, the convention is that a line feed is the new-line character; however, Tandy and Apple computers don't store line feeds as part of the files, so the program must test for carriage return instead.

```
        cpi     alf             ; if (c == '\n')
        cz      incnl           ;    ++nl;
```

Next we test for any of the characters which mark the end of a word (space, tab, carriage return, line feed), and if the current character is one of those, we set inword to NO again and jump back to get the next character.

```
cpi     ' '     ;if (c == ' ' || c == '\n' || c == '\t' || c == 0x0d)
cz      inno    ;inword = NO;
jz      wc3
cpi     alf
cz      inno
jz      wc3
cpi     tab
cz      inno
jz      wc3
cpi     acr
cz      inno
jz      wc3
```

By this stage, the character must be a valid part of a word. If inword is currently NO, then this character is the first one of the word, and so we set inword to YES and count another word. That's the main part of the program done, and it turned out not to be too bad.

```
        lda     inword          ;else if (inword == NO) {
        cpi     no              ;       inword = YES;
        jnz     wc3             ;       ++nw;
        mvi     a,yes           ;}
        sta     inword
        lhld    nw
        inx     h
        shld    nw
        jmp     wc3
```

Now comes the final part of the program, the printing of results. The printing of messages is done by an 'in-line print' routine, which will be explained later. For the moment, just believe that the processor does not try to execute the message text when it returns from the 'ilprt' routine. Then we load HL with the result to be printed, and call a decimal number output routine. Finally we print a carriage return, line feed at the end of the line:

```
wc9:
        call    ilprt           ; printf("number of lines = %u\n",nl);
        db      'number of lines = ',0
        lhld    nl
        call    decout
        call    crlf
        call    ilprt           ; printf("number of words = %u\n",nw);
        db      'number of words = ',0
        lhld    nw
        call    decout
        call    crlf
        call    ilprt           ; printf("number of chars = %u\n",nc);
        db      'number of chars = ',0
        lhld    nc
        call    decout
        call    crlf
        lhld    oldsp           ; restore stack
        sphl
        ret
```

Well, that wasn't so bad, was it? The only problem is, we now have some subroutines to write to perform lower-level tasks for the main program, and structured design techniques don't help quite so much at this level. However, they are generally quite short.

We'll start with the file open subroutine, which simply clears a couple of bytes in the file control block, sets DE to point to the fcb, and then calls the BDOS to perform the appropriate function. The FCB has been initialised by the CCP, so it already contains the file name.

```
;
;       /* subroutines */
;
fopen:                                  ; opens a file named in fcbl
        xra     a
        sta     fcbl + ex               ; zero extent, s2, current record
        sta     fcbl + s2
        sta     fcbl + cr
        lxi     d,fcbl
        mvi     c,openf
        call    bdos
        ret
```

Next comes the trickiest part of the program: the 'get character' function. The trouble is that CP/M wants to read the disk 128 bytes at a time, whereas we only want one. So what we do is make a subroutine that uses a pointer to read successive characters from a 128-byte buffer, and refills that buffer whenever necessary.

Since the default buffer is 128 bytes from 0080H to 00FFH, we can tell the pointer has over-run the buffer end if the least significant byte becomes zero. If we were writing a fully functional 'getc' routine to operate with a buffer anywhere in memory, a slightly more sophisticated technique would be required.

If the pointer has reached the end of the buffer, then we call a routine which fills the buffer. Notice that we have to preset the pointer to force getc to fill the buffer the first time it is called.

```
getc:                           ; gets a character from buffer,
        push    h               ; refills buffer if necessary
        lhld    lastc           ; get pointer into buffer
        inx     h               ; increment it
        shld    lastc           ; and save it again
        mov     a,l             ; have we reached the end of the buffer?
        cpi     0h
        cp      fillbuff        ; if so, then refill it
        mov     a,m             ; get the character
        pop     h
        ret

lastc   dw      defdma + 7fh
```

The fill buffer routine sets DE to point to the FCB and calls the BDOS, then resets the character pointer.

```
fillbuff:
        lxi     d,fcbl          ; point to fcb
        mvi     c,readrec       ; and get record
        call    bdos
        lxi     h,defdma
        shld    lastc           ; reset character pointer to beginning
        ret                     ; of buffer
```

The next routine sets inword to NO:

```
inno:                           ; set inword = no
        push    psw
        mvi     a,no
        sta     inword
        pop     psw
        ret
```

Then comes the routine to increment the number of lines:

```
incnl:                          ; nl = nl + 1
        push    h
        lhld    nl
        inx     h
        shld    nl
        pop     h
        ret
```

The inline print routine is a very handy routine which uses a nice feature of the 8080 family of processors. When the routine is called, the CALL instruction places the return address (the next byte after the CALL) on the stack. This routine swaps the top of the stack with HL and uses it as the address of the string to be printed. When it detects the null (0) byte at the end of the string, it swaps the top of stack with HL again, and returns – only by now HL has been incremented past the string, so the return is to the correct place. It's one of those nice, satisfying, elegant things you can occasionally do in assembler. ▶

2 of 3 YC Nov 83

```
ilprt:                          ; in-line print routine
        xthl                    ; get ptr and save hl
ilplp:
        mov     a,m             ; get char
        ora     a               ; reached end
        jz      ilplx           ; yes, exit
        call    cout            ; print char
        inx     h               ; point to next
        jmp     ilplp           ; and go round
ilplx:  inx     h               ; pt to byte after ending 0
        xthl                    ; restore hl and return
        ret
```

Outputting decimal numbers is a tricky task, since it involves division by 10. Fortunately, we are only dealing with positive numbers, which makes life a bit easier.

This routine saves the processor registers, then performs repeated subtractions of 10 until it sees a negative result, when it adds 10 back in again. The number of times it was able to subtract is the quotient. It then tests for zero remainder, and if the remainder is not zero, calls itself again to output remaining digits of the result, which is why the routine saves the processor registers.

Note also that the routine outputs the digits after returning from itself, which is how it calculates the digits to be output from least significant to the most, but outputs them in reverse order.

```
decout:                         ; decimal output routine
        push    b
        push    d
        push    h
        lxi     b,-10           ; radix for conversion
        lxi     d,-1            ; this becomes no divided by radix
decl:   dad     b               ; subtract 10
        inx     d
        jc      decl
        lxi     b,10
        dad     b               ; add radix back in once
        xchg
        mov     a,h
        ora     l               ; test for zero
        cnz     decout          ; recursive call
        mov     a,e
        adi     '0'             ; convert from binary to ASCII
        mov     e,a             ; to e for output
        mvi     c,conout
        call    bdos
        pop     h
        pop     d
        pop     b
        ret
```

P 104.

Finally, there are a couple of routines to output a CRLF and do console output, followed by the variables. Notice that the initial stack pointer is declared at the end of the stack space, not the start, as the stack grows downwards; this caused me hours of fun once!

```
crlf:                           ; print CRLF
        mvi     a,acr
        call    cout
        mvi     a,alf           ; no call, no ret required

cout:                           ; output character
        push    psw
        push    b
        push    d
        push    h
        mov     e,a
        mvi     c,conout
        call    bdos
        pop     h
        pop     d
        pop     b
        pop     psw
        ret

inword  ds      1               ; inword flag
nl      dw      0               ; number of lines
nw      dw      0               ; number of words
nc      dw      0               ; number of characters
oldsp   ds      2               ; old stack pointer
        ds 256                  ; stack space
stk     equ     $

        end
```

That's it. It's not terribly complex, but it does illustrate a few points about structured design.

If it wasn't for the fact that I had set tab equal to 8, not 9, when I first typed the program in, it would have worked first time, which is unusual for assembler programs. Of course, modules like the 'getc' function were written long ago and had been tested out in other programs, but generally, each of them only had one error to fix at the first testing stage.

The design for getc, by the way, came from the book Software Tools, by Kernighan and Plauger, where it appears in the first chapter, written in FORTRAN. I can't remember where I first saw the decout routine, but I've been using it for years. ❑

your computer tutorial

Understanding Assembler – Part XV

Moving on from writing straight Assembler, Les Bell introduces some higher-level tools for more complex projects.

MUCH AS I LIKE WRITING ASSEMBLER (really?), I do like tools that make it easier. Over the years I have built up quite a library of routines from various sources, ranging from public domain programs, such as are found in the CP/M Users Group, to Scelbi's 8080 Cookbook.

These handy subroutines can either be typed into programs or cut and pasted into place using a good text editor – which is the way I used to do it until I got what is an even more useful tool: a macroassembler. That's not just a big assembler, although it is a bit bigger than the standard CP/M assembler ASM; it's a smarter assembler.

The term macro, in computer science, refers to a text substitution done before assembly, compilation or execution of code. Macros are a form of shorthand; they enable a short word or phrase to stand for a long and possibly complex section of code.

Before the code is assembled (or compiled or whatever), the macro word or phrase is replaced by its full meaning. This is then assembled.

For example, a common requirement in most programs is to print a message at the console. In previous examples in this series, we have simply included a subroutine called ilprt, which printed the message stored after the call instruction which entered it.

On the other hand, we could do the same thing with a macro. At the beginning of the program, we define a macro which contains the subroutine, together with the appropriate call instruction. From then on, we can use the instruction

```
ilprt      'whatever text',0
```

throughout our programs. For those who are curious, and would read ahead, here's the macro, but you'll have to wait a bit for the explanation. It's worth mentioning one point here: the first time the macro is invoked, it generates both the subroutine call and the subroutine itself. Obviously, including the whole subroutine many times in a program (every time we want to print a message) is extremely wasteful, so from then on, it simply generates the call instruction.

```
;Inline print macro
;  print chars ending in 0 pted to by return address; return to byte after
;
print   macro   str
        local   over
        jmp     over        ;; jump over in-line subroutine

ilprtl: xthl                ; get ptr and save hl
;
ilplp:
        mov     a,m         ; get char
        ora     a           ; reached end
        jz      ilplx       ; yes, exit
        call    cout        ; print char
        inx     h           ; point to next
        jmp     ilplp       ; and go round
ilplx:  inx     h           ; pt to byte after ending 0
        xthl                ; restore hl and ret
        ret
;
over:
print   macro   ?s          ;; redefinition
        call    ilprtl
        db      ?s
        db      0
        endm

        print   str
        endm
```

Real Macro Assemblers

There are several macro assemblers on the market for CP/M. The most common is the Digital Research MAC assembler, which operates in much the same way as ASM, except that it can handle macros. In other words, it generates a HEX file, which is then loaded. MAC is an 8080 assembler, though it is supplied with a set of macros which can handle the Z-80 instruction set.

Another popular assembler is Microsoft's MACRO-80, which is a relocating assembler. Rather than generating a HEX file, it produces a REL file, which can then be linked to other REL files to produce the required COM file. This allows programs to be assembled in manageable and independent sections. MACRO-80 has a couple of other useful features: it can assemble Z-80 opcodes, and it can also assemble code which will load in one location for execution at another.

Digital Research's answer to MACRO-80 is RMAC, which is the standard assembler with CP/M Plus. It provides relocation facilities in much the same way as MACRO-80 – in fact their REL files are virtually the same format, but still use fake Z-80 op-codes.

All the above assemblers are modelled on the Intel original macroassembler, and so their instructions are similar. They differ only in the pseudo-ops they use, and so many of my comments about macros apply equally to all the above. There are other assemblers about, most notably the TDL/Xitan assemblers and Sorcim's ACT, but these are different to a considerable extent and so while the general principles apply, the details are different.

The examples which follow are based on MAC and RMAC, since those are the assemblers I use. They should be pretty well OK for the Microsoft assembler too, with only minor changes at the most. TDL and ACT users – you're on your own, I'm afraid.

Incidentally, this seems like a good time to make an impassioned plea to software authors not to write Z-80 code, but to stick to the 8080 subset. Not only is it annoying for those of us who don't run Z-80's to find that a program is unuseable, but the authors are missing out on sales. With the availability of 10 MHz 8085's, coupled with a Godbout dual processor board, the 8085 is a popular chip. Besides, the Z-80's extra instructions rarely produce worthwhile improvements anyway. End of lecture.

New Pseudo-Ops

The MAC and RMAC assemblers contain a couple of addi-

1 of 2.

YC Feb 84.

tional pseudo-ops which ASM does not have. First, there's SET (which ASM does have, but we haven't covered). SET works like EQU, except that attempts to equate the same symbol twice cause an error, whereas a symbol can be SET to several different values throughout an assembly. That's not so important in ASM, but for use in macros, it's well-nigh essential.

The PAGE and TITLE directives control the appearance of the PRN file. PAGE causes a page break, so the printer will skip to the top of the next page. Alternatively, PAGE followed by a number sets the page length. The TITLE pseudo-op allows the user to print the title of the program at the top of each page, thus:

```
TITLE     'Monitor Program V1.1'
```

More important, however, are the three built-in macro instructions REPT, IRP and IRPC.

REPT allows automatic repetition of a sequence of instructions. It takes the following format:

```
REPT expression
statement
   .
   .
statement
ENDM
```

For example, here's a section of code to generate a blank jump table:

```
; generates blank jump table
        org     0100h

njumps  set     5

table:  rept    njumps
        jmp     error
        endm
;
error:
;
        end
```

This is quite simple: the set statement sets the number of jumps to five, then the rept macro generates that many jumps. The PRN file produced by the MAC assembler looks like this:

```
                        ; GENERATES BLANK JUMP TABLE

0100                            ORG     0100H

0005 #                  NJUMPS  SET     5

                        TABLE:  REPT    NJUMPS
                                JMP     ERROR
                                ENDM
0100+C30F01                     JMP     ERROR
0103+C30F01                     JMP     ERROR
0106+C30F01                     JMP     ERROR
0109+C30F01                     JMP     ERROR
010C+C30F01                     JMP     ERROR
                        ;
                        ERROR:
                        ;
                                END
```

Notice that inside the lines generated by the macro, there is a '+' sign between each address and the hex codes. This indicates that the code was generated by macro expansion.

The REPT pseudo-op is fine when you want a number of identical sequences of code. However, you will often want some variations throughout the repeated code, and this can be achieved with the IRP command.

The format of the IRP command is similar to that of REPT:

```
IRPC var,<datalist>
statement
   .
   .
statement
ENDM
```

In this case, one or more of the statements in the macro involve the variable identified in the macro header. Each time the macro is expanded, an item from the datalist is substituted for the variable. For example, here's a more sophisticated jump table:

```
; generates a jump table

bios    equ     0000h
bdos    equ     0005h
tpa     equ     0100h
reboot  equ     0f800h
        org     0100h

        irp     ?d,<bios,bdos,tpa,reboot>
        jmp     ?d
        endm

        end
```

In this case, ?d is the variable which will be used for macro substitution. In other words, the macro processor will keep generating jmp statements as long as it is able to substitute one of the labels bios, bdos, tpa and reboot for ?d. Here's the resulting PRN file:

```
                        ; GENERATES A JUMP TABLE

0000 =                  BIOS    EQU     0000H
0005 =                  BDOS    EQU     0005H
0100 =                  TPA     EQU     0100H
F800 =                  REBOOT  EQU     0F800H

0100                            ORG     0100H

                                IRP     ?D,<BIOS,BDOS,TPA,REBOOT>
                                JMP     ?D
                                ENDM
0100+C30000                     JMP     BIOS
0103+C30500                     JMP     BDOS
0106+C30001                     JMP     TPA
0109+C300F8                     JMP     REBOOT

010C                            END
```

As you can see, the MAC assembler has correctly generated the jump table.

Finally, the IRPC macro works just like IRP, except that it is used to substitute single characters into the macro expansion. Its format is:

```
IRPC var,charlist
statement
   .
   .
statement
ENDM
```

Here's an example which shows how to save the 8080 registers with one instruction:

```
        org     0100h

; save 8080 registers

pushall:
        irpc    reg,bdh
        push    reg
        endm

popper:
        irpc    reg,hdb
        pop     reg
        endm

        end
```

There are two things to notice here. First, I couldn't include the accumulator in this save macro because it is referred to as PSW in a push instruction, so I'd have to use the IRP macro for that. Second, notice that the popper macro substitutes the registers in reverse order, for obvious reasons.

Here's the resulting PRN file:

```
0100                            ORG     0100H

                        ; SAVE 8080 REGISTERS

                        PUSHALL:
                                IRPC    REG,BDH
                                PUSH    REG
                                ENDM
0100+C5                         PUSH    B
0101+D5                         PUSH    D
0102+E5                         PUSH    H

                        POPPER:
                                IRPC    REG,HDB
                                POP     REG
                                ENDM
0103+E1                         POP     H
0104+D1                         POP     D
0105+C1                         POP     B

0106                            END
```

Next month, we'll get on to defining our own macros and looking in depth at parameter substitution. ❑

2 of 2. YC Feb 84

your computer tutorial

Understanding Assembler — Part XVI

Last time we looked at the macro definitions which are built in to the MAC and RMAC assemblers. This month, it's time to investigate how to write your own macros.

THE MOST IMPORTANT application of macros is the definition of your own library of functions. These fall into a number of areas; the most common ones are input/output, operating system calls and expanded language facilities. We'll look at each of these in turn.

Macro substitution involves the replacement of a single-line pseudo-instruction in the original assembly language code with a sequence of instructions which actually perform the desired function. At the simplest level, it's very simple, but when extending its power, the macro facility can be very complex. Needless to say, for both our sakes we're not going to delve into the most advanced aspects of macro systems.

When the assembler encounters the MACRO pseudo-op during its first pass through the program source code, it stores the text it finds after that into its symbol table area, until it finds an ENDM or EXITM pseudo-op. Then, whenever it encounters the macro name in the opcode field, it replaces it with the text in the symbol table. This process is called macro expansion.

For example, we frequently need to save all registers on the stack before calling a subroutine, and it is tempting to try to write a subroutine that will perform this task. However, note that this subroutine will push registers onto the stack and then try to return to an address given by the last register pair to be pushed, so it's not as easy as it first looks (though a solution is possible using the XTHL instruction). It's easier to use a pair of macros for the job.

```
PUSHAL  MACRO
        PUSH    PSW
        PUSH    B
        PUSH    D
        PUSH    H
        ENDM

POPALL  MACRO
        POP     H
        POP     D
        POP     B
        POP     PSW
        ENDM
```

Including these macros in-line where required will do the job.

Another frequent requirement is to use the CP/M BDOS functions to perform tasks such as sending characters or strings to the console, or reading and writing files. As you will recall, this is done by loading the D or DE registers with the data to be output, and the C register with the function number, then calling location 5.

We can use macros to remove much of the tedium of setting up registers, saving prior contents and so on. The simplest way to do this is to create a set of macros, one for each BDOS call. For example, here's a pair of macros to input and output characters through the accumulator:

```
BDOS    EQU     5

CONRD   MACRO
        PUSH    H
        PUSH    D
        PUSH    B
        MVI     C,1
        CALL    BDOS
        POP     B
        POP     D
        POP     H
        ENDM

CONWR   MACRO
        PUSH    H
        PUSH    D
        PUSH    B
        MVI     C,2
        MOV     E,A
        CALL    BDOS
        POP     B
        POP     D
        POP     H
        ENDM
```

Another technique is to use a general BDOS call macro, but the problem here is that different kinds of parameters are used. On some calls (the ones for character I/O to the console and peripherals) a single character is output at a time, while for the disk functions, the parameter passed to the BDOS is generally an address in DE. The solution is to have several different sets of macros; macros for each of the I/O functions and a general macro for disk BDOS functions:

```
RETVERS EQU     12
RESDSK  EQU     13
SELDSK  EQU     14
OPEN    EQU     15
CLOSE   EQU     16
        .
        (etc)
        .
DBDOS   MACRO   FUNC, PARAM
        IF      NOT NUL PARAM
        LXI     D,PARAM
        ENDIF
        MVI     C,FUNC
        CALL    BDOS
        ENDM
```

In this case, I have used the NUL function of MAC to test whether a parameter has been passed to the macro. For example, function 12, get version number, takes no parameters. Therefore, a call to the BDOS to get the CP/M version number would be written

```
DBDOS   RETVERS
```

while the select disk function (14) requires the disk number (A = 0, B = 1, etc) to be placed in E, so it would be written

```
DBDOS   SELDSK,1
```

A more complex case is the printing of a string. Here, we can use the BDOS print string function to get the work done. A simple case is when we want to print an error message.

```
ACR     EQU     13
ALF     EQU     10
BDOS    EQU     5
```

▶

1 of 2.

Apr 84 YC.

```
PRINT   MACRO   MESSAGE
        LOCAL   ?OVER,?MSG
        JMP     ?OVER

?MSG    DB      MESSAGE
        DB      ACR,ALF
        DB      '$'

?OVER   LXI     D,?MSG
        MVI     C,9
        CALL    BDOS
        ENDM
```

This example introduces a couple of complexities. Note first, that the generated code must include the text of the message, so it must include a jump to get around the text. We can't simply come up with a label for the purpose, as the next time the macro is invoked in the program the assembler will tell us we've already used the label.

Instead, we define a couple of local labels, ?OVER and ?MSG: one to be the target of the jump over the text, and the other to be the address of the text itself. When the macro is expanded, the assembler will supply its own labels, allowing the macro to be re-used elsewhere, as this listing shows:

```
000D =          ACR     EQU     13
000A =          ALF     EQU     10
0005 =          BDOS    EQU     5

                PRINT   MACRO   MESSAGE
                        LOCAL   ?OVER,?MSG
                        JMP     ?OVER
                ?MSG    DB      MESSAGE
                        DB      ACR,ALF
                        DB      '$'
                ?OVER   LXI     D,?MSG
                        MVI     C,9
                        CALL    BDOS
                        ENDM

                        print   'Now is the time'
0000+C31500             JMP     ??0001
0003+4E6F772069??0002   DB      'Now is the time'
0012+0D0A               DB      ACR,ALF
0014+24                 DB      '$'
0015+110300     ??0001  LXI     D,??0002
0018+0E09               MVI     C,9
001A+CD0500             CALL    BDOS
001D C30000             jmp     0

0020                    end
```

The labels ?OVER and ?MSG are replaced by ??0001 and ??0002 respectively. As the assembler encounters more local labels, it will generate more labels of that kind.

There is still another problem with macros of this kind – they generate in-line code each time they are invoked. This means that large chunks of code are repeated in each macro, when they could be more efficiently used as subroutines. Is there a way of turning macros into subroutines? The answer is, obviously, yes – otherwise I wouldn't have mentioned it!

As a macro is expanded, the assembler follows the normal sequence of events: placing op-codes into the source text and assembling them, and executing pseudo-ops. These pseudo-ops include macro definition statements, like MACRO and ENDM. Because of this serendipitous operation, we can use macros to redefine themselves.

It works like this. The first time a macro is expanded, we turn it into a subroutine, a jump past that subroutine, and a call to the subroutine. Once the subroutine is in the program, we can simply call it, so we redefine the macro as a simple subroutine call. For example, consider the in-line print subroutine we used in the monitor program some time ago. That could be replaced by a macro in a macro library. Here's the code:

```
;Inline print macro
;  print chars ending in 0 pted to by return address; return to byte after
;
print   macro   str
        local   over
        jmp     over        ;; jump over in-line subroutine

ilprtl: xthl                ; get ptr and save hl
;
ilplp:
        mov     a,m         ; get char
        ora     a           ; reached end
        jz      ilplx       ; yes, exit
        call    cout        ; print char
        inx     h           ; point to next
        jmp     ilplp       ; and go round
ilplx:  inx     h           ; pt to byte after ending 0
        xthl                ; restore hl and ret
        ret
;
over:
print   macro   ?s          ;; redefinition
        call    ilprtl
        db      ?s
        db      0
        endm

        print   str
        endm
```

Here's how it works: first we define the local variable over and insert a jump to it. This jumps around the subroutine. Then comes the in-line print subroutine itself. Once the subroutine has safely been incorporated in the generated code, we then redefine the macro as a call to the subroutine. Finally, before finishing the original macro expansion, we insert an invocation of the new macro – in other words, a subroutine call to ilprt1.

Defining macros in this way makes it possible to have libraries of macros and simply stick them into a program by name; the first time the assembler encounters the macro, it sticks in the appropriate subroutine, but thereafter it only generates subroutine calls.

It's fair to say that macro substitution, using these kinds of tricks, can be pretty mind-boggling, so if you're still with us, well done! However, bear in mind that there is a lot more to the use of macros than this, especially when you start to delve into recursive macro expansions and other *recherché* stuff.

The construction of macro libraries, as discussed above, is very easy – at least it is with MAC and RMAC. Simply collect all your macros into a text file, and call it MACROS.LIB or similar. Then, at the top of your program, insert the line MACLIB MACROS, and all your macro definitions will be dragged in, ready for use. For example, suppose I put the in-line print macro into a library of its own, called ILPRT.LIB. Here's a simple example which demonstrates how the MACLIB command works, and also proves that macro redefinition really does work:

```
        maclib  ilprt
bdos    equ     5

        print   'Now is the time '
        print   'for all good men'
        jmp     0

cout:   push    h
        mov     e,a
        mvi     c,2
        call    bdos
        pop     h
        ret
        end
```

Here's the PRN file which shows how it works:

```
                        maclib  ilprt
0005 =          bdos    equ     5

                        print   'Now is the time '
0000+C31300             JMP     ??0001
0003+E3         ILPRTl: XTHL            ; GET PTR AND SAVE HL
0004+7E                 MOV     A,M     ; GET CHAR
0005+B7                 ORA     A       ; REACHED END
0006+CA1000             JZ      ILPLX   ; YES, EXIT
0009+CD3E00             CALL    COUT    ; PRINT CHAR
000C+23                 INX     H       ; POINT TO NEXT
000D+C30400             JMP     ILPLP   ; AND GO ROUND
0010+23         ILPLX:  INX     H       ; PT TO BYTE AFTER ENDING 0
0011+E3                 XTHL            ; RESTORE HL AND RET
0012+C9                 RET
0013+CD0300             CALL    ILPRTl
0016+4E6F772069         DB      'Now is the time '
0026+00                 DB      0
                        print   'for all good men'
0027+CD0300             CALL    ILPRTl
002A+666F722061         DB      'for all good men'
003A+00                 DB      0
003B C30000             jmp     0

003E E5         cout:   push    h
003F 5F                 mov     e,a
0040 0E02               mvi     c,2
0042 CD0500             call    bdos
0045 E1                 pop     h
0046 C9                 ret
0047                    end
```

The art of writing macros using redefinition and other tricks is not exactly dying out, but it's under a lot of pressure from a simpler alternative: relocating assemblers and linking loaders. I'll discuss them in the next article.

2 of 2 YC Apr 84

PE micro-file

R.W.Coles

FILESHEET 1 8080A · 8085A

THE Intel 8080 8 bit NMOS microprocessor first appeared in 1973 as a successor to the more limited 8008 PMOS device. The 8080A was the first microprocessor to capture the imagination of designers and was a fundamental cog in the microprocessor revolution generating annual sales of over 2 million devices per year in its heyday. The success of this chip resulted in the spawning of two, more powerful successors, the Z80 from Zilog which had an enhanced instruction set but basically the same bus configuration, and the 8085A from Intel which had basically the same instruction set but a new multiplexed bus structure. Both of the newcomers appeared in 1977 and have now replaced the 8080A for all new applications with the Z80 being most popular for data processing and the 8085 being more successful as a controller.

In order to squeeze the maximum performance from the NMOS technology available in the early 1970s the 8080A was designed to use three supply rails of +5, −5 and +12 volts and had to have two additional support chips to provide clock generation and bus interface. The main competition to the 8080A in the early days was the Motorola 6800 which despite using only two chips and a single supply voltage was never as popular due to its lower overall performance.

The 8080A has a common instruction and data memory space of 64 kilobytes and a separate I/O space of 256 ports which, together with a good general purpose instruction set, made it useful for a wide range of applications in control and data processing.

The 8085 was an attempt by Intel to maintain the sales momentum created by the 8080A, although it could be argued that the competing Z80 from Zilog did a better job. The 8085 needs no support chips except for memory and I/O, and will run faster than the 8080A from a single 5V supply. To free extra interface pins the 8085A has a multiplexed data and address bus with the new connections being used for extra interrupts and serial I/O in addition to the necessary control and clock lines. Introduced at the same time as the 8085A were two special peripheral devices also in 40 pin packages. The 8155 provides 256 bytes of RAM, 22 parallel I/O lines and a 14 bit timer while the 8355 provides 2K bytes of ROM and 16 parallel I/O lines. Using the 8085A with these two peripherals it is possible to build a powerful processor system with RAM, ROM and comprehensive I/O using just three chips.

REGISTERS

The 8080A and the 8085A have an identical data register arrangement although the 8085A does have an additional register which is used in the control of its extra serial I/O and Interrupt lines. Both devices have eight addressable 8 bit registers which can be used as four 16 bit register pairs for many operations. Perhaps most important of these is the 8 bit Accumulator register which is the implied focus of many instructions including the memory reference, arithmetic, and I/O groups. For some operations this register is paired with the flag register which itself provides single bit status information about data in the accumulator after arithmetic and logical operations. Flag bits are provided to report on five possible status conditions as shown on the file sheet, with the remaining 3 bits being unused. The BC, DE and HL registers are essentially general purpose in nature and can be used as temporary storage for 8 and 16 bit data values, as 8 and 16 bit counters, or as 16 bit memory address pointers. The HL register is particularly important as a memory pointer since it is used by a number of memory reference instructions. It is also used as an "accumulator" for 16 bit arithmetic. A smaller number of instructions use the BC and DE pairs as pointers, and either of these register pairs can be added to the HL pair to give a limited 16 bit arithmetic capability.

In addition to the four register pairs already discussed there are two other 16 bit registers which have dedicated functions. The Program Counter register always points to the next instruction to be executed and therefore contains a 16 bit address. The Stack Pointer always points to the top of the last-in-first-out stack area maintained in read/write memory for the storage of subroutine return addresses and register values saved during interrupts or for other purposes. The Stack Pointer is decremented each time data is "pushed" on to the stack and is incremented each time data is "popped" off the stack.

The generous register set of the 8080 was one of the reasons for its success over the Motorola 6800, but the specialised uses of the BC, DE and HL pairs also had the effect of producing a less regular and "messy" instruction set making it necessary for the programmer to remember just what particular pairs can and cannot be used for. The more modern 16 bit processors overcome this problem by making their registers completely general purpose and non-specialised wherever possible. Lacking in the 8080/8085 is the useful feature of an index register such as that provided by the 6800, although this job can be performed by the register pairs at the cost of using extra instructions.

INSTRUCTION SET

As mentioned above, the 8080/8085 instruction set is rather "messy" due to the somewhat specialised nature of the large register array, but this does make these devices very powerful considering their small chip areas. The 78 basic instructions of the 8080 are used to move data between registers, between a register and memory, between a register and an I/O port, and to carry out arithmetic and logical operations. Instructions are also included to perform conditional and unconditional jumps and to control processor operation. Two additional 8085A instructions, RIM and SIM, are ingeniously used to provide access to, and control over, the extra serial I/O and interrupt features not present on the 8080.

A comprehensive array of arithmetic and logical operations are provided including 8 and 16 bit binary addition, 8 bit binary subtraction, binary coded decimal (BCD) arithmetic on packed BCD values, logical operations such as AND, OR, XOR and Compare, and a range of accumulator shifts and carry flag modifiers. One item missing from this group is the ability to set, test, and reset, individual accumulator bits which is a very useful feature for control applications. These operations can be performed by shifting the relevant bit into the carry flip-flop or by using logic instructions, however.

Four addressing modes are used as follows:—Direct, in which a memory address is specified as part of the instruction; Register, in which a register or register pair is specified; Register Indirect, in which the instruction specifies a register pair which itself contains a memory address; and Immediate, in which the instruction contains not a reference to a data area but the actual data itself. One particularly useful feature of the instruction set is the provision of a group of eight Restart instructions which cause an immediate jump to fixed vectors in low memory. These instructions use only a single byte and are used for hardware interrupt service or as software interrupts. Access to the separate I/O address space of 256 input and 256 output ports is provided by means of the instructions IN and OUT which are fast because they are only 2 bytes long. The separate I/O address space is useful because it does not encroach on main memory, but it is still possible to use memory mapping for I/O ports if required for a simple system not needing the full 64K memory address range.

8080A/8085A REFERENCE FILE SHEET

GENERAL

The 8080A was the first of the mid-range NMOS 8 bit processors and is certainly the most widely used. It has a good general purpose architecture and is very well supported with both hardware and software. The 8085A has essentially the same instruction set as the 8080A but needs only a 5V supply and has many additional features such as on-chip clock, serial I/O and four new interrupt lines. Extra pins for these functions have been made available by multiplexing the low order address bits with the data bus. A complete 8085A system with 2K bytes of ROM, 256 bytes of RAM, a timer and 38 I/O lines can be built with just three 40 pin chips by utilising the 8355 (ROM I/O) and the 8155 (RAM I/O TIMER) combination devices.

REGISTERS

REGISTERS: The 8080/8085 has seven 8 bit general purpose registers. Six of these can be addressed as the three 16 bit pairs BC, DE, HL.

Notes
1) PSW = Processor Status Word
2) HL is used as memory pointer for register indirect addressing.

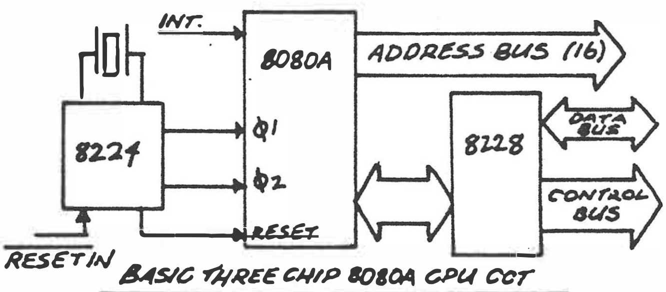
ACCUM	8	FLAGS	6	PSW
B	8	C	8	BC
D	8	E	8	DE
H	8	L	8	HL
STACK POINTER			16	SP
PROGRAM COUNTER			16	PC

FLAGS:

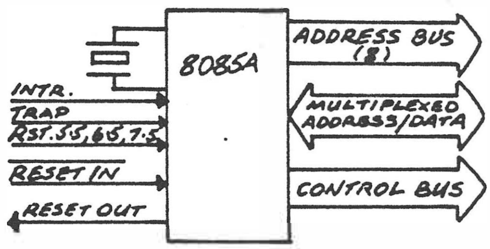
D7	D6	D5	D4	D3	D2	D1	D0
SIGN	ZERO		AUX CARRY		PARITY		CARRY

INSTRUCTION SET AND SOFTWARE

The 8080 has 78 basic instructions and the 8085 has two more, RIM and SIM which support the additional interrupts and serial I/O. One, two and three byte instructions are used and Direct, Register, Register Indirect and Immediate addressing modes are available. Full binary and BCD arithmetic is possible on 8 bit bytes, and some 16 bit arithmetic is possible using the HL pair as an accumulator. A separate address space is available for I/O using the IN and OUT instructions. Very well supported with software including Tiny Basics and the CPM operating system.



BASIC THREE CHIP 8080A CPU CCT



BASIC SINGLE CHIP 8085A EQUIVALENT

PERFORMANCE DATA

PERFORMANCE DATA	8080A	8085A
MEMORY ADDRESS RANGE:	64K	64K
I/O ADDRESS RANGE:	256	256
CLOCK FREQUENCY:*	2MHz	3·125
POWER SUPPLIES:	+5, -5, +12V	+5V
INTERRUPTS:	INT.	INTR. TRAP RST 5.5 RST 6.6 RST 7.5

* NOTE: HIGH SPEED VERSIONS OF 8080A (3MHz) AND 8085A (5MHz) ALSO AVAILABLE

BENCHMARKS	8080A	8085A
ADD REGISTER TO ACCUM:	2μS	1·28μS
OUTPUT ACCUMULATOR TO PORT:	5μS	3·2μS
MOVE FROM MEMORY TO MEMORY:	8μS	5·12μS

8080A

A10	1	40	A11
GND	2	39	A14
D4	3	38	A13
D5	4	37	A12
D6	5	36	A15
D7	6	35	A9
D3	7	34	A8
D2	8	33	A7
D1	9	32	A6
D0	10	31	A5
-5V	11	30	A4
RESET	12	29	A3
HOLD	13	28	+12V
INT	14	27	A2
Ø2	16	26	A1
INTE	16	25	A0
DBIN	17	24	WAIT
WR	18	23	READY
SYNC	19	22	Φ1
+5V	20	21	HLDA

8085A

X1	1	40	VCC
X2	2	39	HOLD
RESET OUT	3	38	HLDA
SOD	4	37	CLK OUT
SID	5	36	RESET IN
TRAP	6	35	READY
RST 7.5	7	34	IO/M
RST 6·5	8	33	S1
RST 5·5	9	32	RD
INTR.	10	31	WR
INTA	11	30	ALE
AD0	12	29	S0
AD1	13	28	A15
AD2	14	27	A14
AD3	15	26	A13
AD4	16	25	A12
AD5	17	24	A11
AD6	18	23	A10
AD7	19	22	A9
VSS	20	21	A8

PIN DIAGRAMS

MANUFACTURERS

ORIGINATOR — INTEL

2nd Sources 8080A: SIEMENS, AMD, NEC. NATIONAL, SIGNETICS, HITACHI

2nd. Sources 8085A: AMD, SIEMENS, NEC

SUPPORT CHIPS

8080A Needs 8224 and 8228 and has a large family of support devices including: 8251 (USART), 8255 (Parallel I/O) 8253 (Timer) 8259 (Interrupt Control) 8257 (DMA)
8085A has two special combination I/O memory chips 8355 and 8155 in addition to above devices.

TABLE 8. INSTRUCTION SET SUMMARY

Mnemonic	Description	Instruction Code[1] $D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0$	Clock[2] Cycles
MOVE, LOAD, AND STORE			
MOVr1 r2	Move register to register	0 1 D D D S S S	4
MOV M r	Move register to memory	0 1 1 1 0 S S S	7
MOV r M	Move memory to register	0 1 D D D 1 1 0	7
MVI r	Move immediate register	0 0 D D D 1 1 0	7
MVI M	Move immediate memory	0 0 1 1 0 1 1 0	10
LXI B	Load immediate register Pair B & C	0 0 0 0 0 0 0 1	10
LXI D	Load immediate register Pair D & E	0 0 0 1 0 0 0 1	10
LXI H	Load immediate register Pair H & L	0 0 1 0 0 0 0 1	10
LXI SP	Load immediate stack pointer	0 0 1 1 0 0 0 1	10
STAX B	Store A indirect	0 0 0 0 0 0 1 0	7
STAX D	Store A indirect	0 0 0 1 0 0 1 0	7
LDAX B	Load A indirect	0 0 0 0 1 0 1 0	7
LDAX D	Load A indirect	0 0 0 1 1 0 1 0	7
STA	Store A direct	0 0 1 1 0 0 1 0	13
LDA	Load A direct	0 0 1 1 1 0 1 0	13
SHLD	Store H & L direct	0 0 1 0 0 0 1 0	16
LHLD	Load H & L direct	0 0 1 0 1 0 1 0	16
XCHG	Exchange D & E. H & L Registers	1 1 1 0 1 0 1 1	4
STACK OPS			
PUSH B	Push register Pair B & C on stack	1 1 0 0 0 1 0 1	12
PUSH D	Push register Pair D & E on stack	1 1 0 1 0 1 0 1	12
PUSH H	Push register Pair H & L on stack	1 1 1 0 0 1 0 1	12
PUSH PSW	Push A and Flags on stack	1 1 1 1 0 1 0 1	12
POP B	Pop register Pair B & C off stack	1 1 0 0 0 0 0 1	10
POP D	Pop register Pair D & E off stack	1 1 0 1 0 0 0 1	10
POP H	Pop register Pair H & L off stack	1 1 1 0 0 0 0 1	10
POP PSW	Pop A and Flags off stack	1 1 1 1 0 0 0 1	10
XTHL	Exchange top of stack. H & L	1 1 1 0 0 0 1 1	16
SPHL	H & L to stack pointer	1 1 1 1 1 0 0 1	6
JUMP			
JMP	Jump unconditional	1 1 0 0 0 0 1 1	10
JC	Jump on carry	1 1 0 1 1 0 1 0	7/10
JNC	Jump on no carry	1 1 0 1 0 0 1 0	7/10
JZ	Jump on zero	1 1 0 0 1 0 1 0	7/10
JNZ	Jump on no zero	1 1 0 0 0 0 1 0	7/10
JP	Jump on positive	1 1 1 1 0 0 1 0	7/10
JM	Jump on minus	1 1 1 1 1 0 1 0	7/10
JPE	Jump on parity even	1 1 1 0 1 0 1 0	7/10
JPO	Jump on parity odd	1 1 1 0 0 0 1 0	7/10
PCHL	H & L to program counter	1 1 1 0 1 0 0 1	6
CALL			
CALL	Call unconditional	1 1 0 0 1 1 0 1	18
CC	Call on carry	1 1 0 1 1 1 0 0	9/18
CNC	Call on no carry	1 1 0 1 0 1 0 0	9/18
CZ	Call on zero	1 1 0 0 1 1 0 0	9/18
CNZ	Call on no zero	1 1 0 0 0 1 0 0	9/18
CP	Call on positive	1 1 1 1 0 1 0 0	9/18
CM	Call on minus	1 1 1 1 1 1 0 0	9/18

Mnemonic	Description	Instruction Code[1] $D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0$	Clock[2] Cycles
CPE	Call on parity even	1 1 1 0 1 1 0 0	9/18
CPO	Call on parity odd	1 1 1 0 0 1 0 0	9/18
RETURN			
RET	Return	1 1 0 0 1 0 0 1	10
RC	Return on carry	1 1 0 1 1 0 0 0	6/12
RNC	Return on no carry	1 1 0 1 0 0 0 0	6/12
RZ	Return on zero	1 1 0 0 1 0 0 0	6/12
RNZ	Return on no zero	1 1 0 0 0 0 0 0	6/12
RP	Return on positive	1 1 1 1 0 0 0 0	6/12
RM	Return on minus	1 1 1 1 1 0 0 0	6/12
RPE	Return on parity even	1 1 1 0 1 0 0 0	6/12
RPO	Return on parity odd	1 1 1 0 0 0 0 0	6/12
RESTART			
RST	Restart	1 1 A A A 1 1 1	12
INPUT/OUTPUT			
IN	Input	1 1 0 1 1 0 1 1	10
OUT	Output	1 1 0 1 0 0 1 1	10
INCREMENT AND DECREMENT			
INR r	Increment register	0 0 D D D 1 0 0	4
DCR r	Decrement register	0 0 D D D 1 0 1	4
INR M	Increment memory	0 0 1 1 0 1 0 0	10
DCR M	Decrement memory	0 0 1 1 0 1 0 1	10
INX B	Increment B & C registers	0 0 0 0 0 0 1 1	6
INX D	Increment D & E registers	0 0 0 1 0 0 1 1	6
INX H	Increment H & L registers	0 0 1 0 0 0 1 1	6
INX SP	Increment stack pointer	0 0 1 1 0 0 1 1	6
DCX B	Decrement B & C	0 0 0 0 1 0 1 1	6
DCX D	Decrement D & E	0 0 0 1 1 0 1 1	6
DCX H	Decrement H & L	0 0 1 0 1 0 1 1	6
DCX SP	Decrement stack pointer	0 0 1 1 1 0 1 1	6
ADD			
ADD r	Add register to A	1 0 0 0 0 S S S	4
ADC r	Add register to A with carry	1 0 0 0 1 S S S	4
ADD M	Add memory to A	1 0 0 0 0 1 1 0	7
ADC M	Add memory to A with carry	1 0 0 0 1 1 1 0	7
ADI	Add immediate to A	1 1 0 0 0 1 1 0	7
ACI	Add immediate to A with carry	1 1 0 0 1 1 1 0	7
DAD B	Add B & C to H & L	0 0 0 0 1 0 0 1	10
DAD D	Add D & E to H & L	0 0 0 1 1 0 0 1	10
DAD H	Add H & L to H & L	0 0 1 0 1 0 0 1	10
DAD SP	Add stack pointer to H & L	0 0 1 1 1 0 0 1	10
SUBTRACT			
SUB r	Subtract register from A	1 0 0 1 0 S S S	4
SBB r	Subtract register from A with borrow	1 0 0 1 1 S S S	4
SUB M	Subtract memory from A	1 0 0 1 0 1 1 0	7
SBB M	Subtract memory from A with borrow	1 0 0 1 1 1 1 0	7
SUI	Subtract immediate from A	1 1 0 1 0 1 1 0	7
SBI	Subtract immediate from A with borrow	1 1 0 1 1 1 1 0	7

TABLE 8. INSTRUCTION SET SUMMARY (Continued)

Mnemonic	Description	Instruction Code[1] $D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0$	Clock[2] Cycles
LOGICAL			
ANA r	And register with A	1 0 1 0 0 S S S	4
XRA r	Exclusive Or register with A	1 0 1 0 1 S S S	4
ORA r	Or register with A	1 0 1 1 0 S S S	4
CMP r	Compare register with A	1 0 1 1 1 S S S	4
ANA M	And memory with A	1 0 1 0 0 1 1 0	7
XRA M	Exclusive Or memory with A	1 0 1 0 1 1 1 0	7
ORA M	Or memory with A	1 0 1 1 0 1 1 0	7
CMP M	Compare memory with A	1 0 1 1 1 1 1 0	7
ANI	And immediate with A	1 1 1 0 0 1 1 0	7
XRI	Exclusive Or immediate with A	1 1 1 0 1 1 1 0	7
ORI	Or immediate with A	1 1 1 1 0 1 1 0	7
CPI	Compare immediate with A	1 1 1 1 1 1 1 0	7
ROTATE			
RLC	Rotate A left	0 0 0 0 0 1 1 1	4
RRC	Rotate A right	0 0 0 0 1 1 1 1	4

Mnemonic	Description	Instruction Code[1] $D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0$	Clock[2] Cycles
RAL	Rotate A left through carry	0 0 0 1 0 1 1 1	4
RAR	Rotate A right through carry	0 0 0 1 1 1 1 1	4
SPECIALS			
CMA	Complement A	0 0 1 0 1 1 1 1	4
STC	Set carry	0 0 1 1 0 1 1 1	4
CMC	Complement carry	0 0 1 1 1 1 1 1	4
DAA	Decimal adjust A	0 0 1 0 0 1 1 1	4
CONTROL			
EI	Enable interrupts	1 1 1 1 1 0 1 1	4
DI	Disable Interrupt	1 1 1 1 0 0 1 1	4
NOP	No-operation	0 0 0 0 0 0 0 0	4
HLT	Halt	0 1 1 1 0 1 1 0	5
EXTRA 8085A INSTRUCTIONS			
RIM	Read Interrupt Mask	0 0 1 0 0 0 0 0	4
SIM	Set Interrupt Mask	0 0 1 1 0 0 0 0	4

NOTES: 1. DDD or SSS: B=000. C 001. D 010. E 011. H 100. L 101. Memory 110. A 111.
2. Two possible cycle times. (6/12) indicate instruction cycles dependent on condition flags.

SOFTWARE

The 8080/8085 family is probably better supported in software than any of the other microprocessors. There is so much software available that it would be quite impossible to list it all. The key to 8080/8085 software is the CP/M disc operating system produced by Digital Research of Pacific Grove, California. Since its introduction, CP/M has become the standard microprocessor operating system and has therefore encouraged large numbers of software writers to produce Interpreters, Compilers, Word processors, games, and utilities. CP/M itself is quite basic but is written in 8080 code so that it is directly compatible with 8080, 8085 and Z80 based systems. So popular is it, that personal computers based on other processors, such as the Apple which uses a 6502, are often upgraded to CP/M compatibility by the addition of an extra 8080 or Z80 processor card so that access to CP/M compatible software is possible.

Of course, not all systems can use discs, and in this case standalone software is desirable. Software distribution is more difficult in this case, but a number of 8080/8085 Tiny Basic Interpreters have been published and there are several books with software listings available. I can recommend the inexpensive Scelbi books which give listings for an 8080 Monitor, Editor, and Assembler.

INTERFACING

The 8080A and 8085A interface to both memory and I/O devices by means of READ and WRITE machine cycles which each have an associated control line output ($\overline{RD}$ and $\overline{WR}$ respectively). An additional control line IO/$\overline{M}$ informs bus users whether the cycle applies to a memory or an I/O device. The main difference between the two processors is the multiplexed bus structure of the 8085A where the eight low order address bits (AO–A7) share the same pins as the data bus and are therefore labelled ADO–AD7. The special purpose 8085A interface chips, the 8155 RAM/IO/TIMER and the 8355 ROM/IO, have internal demultiplexing circuitry so that they can work directly from the 8085 bus. Other devices including general purpose ROM and RAM chips, and interface chips such as the UART, need a non-multiplexed bus and this can be easily achieved by using an external 8 bit latch such as the 74LS373. The 8085A provides a special signal, ALE, to cause the low address data to be latched. With this latch in use, the bus structures of the 8080A and 8085A are virtually identical.

The most versatile interrupt line, INT on the 8080A and INTR on the 8085A can cause a vector to any location in memory with the use of external hardware to force a CALL (Jump to subroutine) instruction on to the bus. This three byte instruction is best generated by the 8259A interrupt controller which will provide separate interrupt vectors for up to eight interrupts. A much simpler scheme can also be used to generate single byte RESTART instructions instead, but of course these vector to fixed locations in low memory. In addition to this general purpose interrupt, the 8085A has four additional fixed vector interrupt lines which do not need any external hardware support. These inputs, RST 5.5, RST 6.5, RST 7.5 and TRAP, cause the processor to vector to locations in low memory positioned between the RESTART vectors which remain available. The TRAP interrupt puts right one criticism of the 8080A by providing a non-maskable interrupt which cannot be ignored. This is useful for important occurrences such as power failure.

One major strength of the 8080A/8085A family is the very wide range of directly compatible interface devices available. In addition to the 8259A Interrupt controller there is the 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART), the 8255A Programmable Parallel Interface (PPI), the 8271 Floppy Disc Controller, the 8278 Programmable Keyboard Interface and many, many more, including devices made for this family by other manufacturers such as N.E.C. Both processors are compatible with a wide range of standard memory components including static and dynamic RAM, ROM, EPROM, and EEPROM.

APPLICATIONS

Unless you are an existing 8080A fan, there would seem to be little point in using this processor for new applications since both the Z80 and the 8085A are actually cheaper and, of course, more powerful. The 8085A still has a part to play in controller applications which can make good use of its extra Interrupts, Serial I/O lines, and the useful 8155A peripheral device, but it is really best suited to applications which are too "big" for one of the single chip processors like the 8748, but not so big that they need one of the newer 16 bit devices. For data processing applications the Zilog Z80 is probably a better choice. Perhaps the main obstacle to using the 8085A in home projects is the inability to use the 8355A masked ROM and I/O device and the consequent need to use a standard EPROM such as the 2716 which therefore makes the use of a bus demultiplexer latch necessary.

THE OBSOLESCENCE PROBLEM

One thorny problem for any budding designer is the very rapid progress in microprocessor technology which produces better, faster, and above all *cheaper* devices at a breakneck pace. There is therefore the ever present spectre of starting a project and then finding that before is is finished a new device has emerged which would do the job better and at a lower cost. This is especially true in the data processing field where development periods tend to be longer.

To avoid the worst of this problem, it is obviously necessary to choose a device which is not about to be superseded. Beware the bargain offer of a wheelbarrow full of National SC/MPs or Intel 8008s for a "Tenner!" At the same time it *is* necessary to choose a device which has been in-play for a sufficiently long period to establish its popularity and which can therefore be expected to have good support and a long life. You can expect the manufacturers to develop their success with popular chips by bringing out improved versions, and this can be an advantage because your "learning" investment can be put to good use on future projects using the enhanced devices when they are available. It is also necessary to remember that, say, a central heating controller may be required to operate for 20 years or more while the lifetime of the majority of microprocessors can be expected to be less than ten years—so remember to buy a spare!

SUPPORT DEVICES

If there *were* any such thing as a typical microprocessor system then in addition to the processor device itself we could expect to find RAM and ROM memory, a parallel I/O port, a serial I/O port, and at least one "special" device such as a disc controller, a maths chip, or an analogue to digital converter. Support devices are available to fill all these requirements and many more besides, and these have to be given serious consideration since they contribute almost as much as the processor itself to the success of any project.

Support devices can be part of a particular microprocessor "family" and these often have special features to simplify their use with that family. Also available are many general purpose devices which can be interfaced to most processors with the addition of a small amount of external logic. All have their part to play. The trend in support devices is towards complex and powerful chips which give a considerable boost to the basic performance of any processor by unloading from it a lot of the system "chores" which it would otherwise have to perform for itself. Prime examples here are the maths processor chips which give systems easy access to floating point arithmetic and high level math functions such as square roots and sines which would normally have to be provided by software routines. Many support devices rival the microprocessors which they serve in chip complexity, and so it is important not to underestimate the task of learning how to initialise and program these devices to perform the required function. Some support chips even have user manuals as thick as those of their attendant microprocessor!

MICRO-FILE FORMAT

Having set the scene, and perhaps frightened, but hopefully inspired many readers, we can now return to how the MICRO-FILE series has been designed to help!

To make any kind of objective assessment of a number of microprocessor devices it is normally necessary to purchase the relevant manuals, and these are not cheap. Having purchased the manuals, a period of intensive study is required to sort out the important characteristics and to come to any conclusion. Remember too, that the manuals are written by the manufacturer and are therefore unlikely to point out any shortcomings!

MICRO-FILE builds up month by month to provide a complete quick reference guide to the more popular microprocessors. Each MICRO-FILE entry consists of a quick reference fact sheet, designed for easy filing, and explanatory text which provides further information and application data. The sheets can be removed from the magazine and placed in a binder for filing.

This introductory article can form the binding "covers". At present there are plans to include about twelve of the most popular processors, but this may be extended later if necessary. So if you collect the whole series it will form a 48 page (or more) reference book on microprocessors plus this "cover" section.

The first FILESHEET considers the Intel 8080A and its successor the 8085A, two of the most popular processors so far, with the 8080A often considered to be the processor which really started the microprocessor revolution.

The reference fact sheet is intended to provide all the essential information about a processor or a processor family, including general background details, register arrangement, instruction set and software, system schematics, performance data, pin connections and basic support chip information. Using these sheets it will be possible to compare processors and to choose the best one for a particular application. Readers not interested in go-it-alone projects can use the sheets to assess the potential power of readily built systems using a particular processor, to help with system trouble shooting and interfacing, or simply to improve their knowledge of the subject.

5 of 5

PE micro-file R.W.Coles
FILESHEET 3 Z80

WHEN the Z80 8 bit NMOS device was introduced by Zilog in 1977 it immediately set the microprocessor world buzzing because it offered so many powerful new features in such an easy to use package. When I first read the specification of the chip early in the launch year I remember thinking "The Z80 is great, but Zilog won't be able to manufacture and sell it at a reasonable price for years!" I was wrong. By the end of 1977 the Z80 was a practical reality and sales were starting to take off like a sky-rocket.

There was plenty for everyone to get excited about, because the Z80 was designed from the outset to be bigger and better than anything else but especially the Intel 8080A which was the market leader at that time. The Zilog Corporation was actually founded on the Z80 project by a group of ex-Intel engineers who *knew* they could produce a super-micro if *only* someone would let them. Intel wouldn't, perhaps because of their 8080A sales success and their own rather tame 8085A plans, and so Zilog and the Z80 were born.

To guarantee their fledgling a good start, they decided that it had to be compatible with the 8080A to gain a ready acceptance by those who had already become 8080A orientated, and to ensure success they decided that the Z80 would have to be able to do everything that anyone could wish from an 8 bit device. It had to run from a single 5V supply (like the 6800), it had to use only one chip for the CPU group (as opposed to three and two for the 8080A and 6800 respectively), it had to be faster than its competitors, offer sophisticated, mini-computer style I/O and interrupts, and above all it had to have a large instruction set with facilities for indexed addressing, relative jumps, bit manipulation, 4 bit nibble manipulation, extended 16 bit arithmetic, and "macro" instructions for the manipulation of whole blocks of data. A recipe which made the new chip equally at home in data processing or controller applications and with a competitive edge which left the opposition standing!

Despite this blockbuster approach to excellence, the Z80 did have its critics almost from the beginning, especially those who thought that the price paid for 8080A compatibility was too high, and that a fresh architecture and instruction set which was less complicated and "messy" would have been better. With the benefit of hindsight it is probably correct to say that its 8080A compatibility won the Z80 a bigger following than a new architecture might have done, but it is also certainly true that many designers have thrown up their hands in despair at some of the more illogical idiosyncrasies resulting from that compatibility! For my part, I consider the Z80 much easier to use and to program when its ancestry is properly understood, and for this reason all readers are urged to study the 8080A/8085A Microfile article before attempting to come to grips with the mighty Z80!

Since 1977 the Z80 has remained virtually unchallenged as king of the 8 bit chips except for a creditable (but late) contender from Motorola in the shape of their classy 6809. Of course there are many more powerful 16 bit devices on the market today, but the Z80 is by no means obsolete since it will have a large price advantage over these newer chips for the foreseeable future. Even when the Z80 *does* become a geriatric micro, it seems that some of its idiosyncrasies will live on. Zilog have recently announced the compatible Z800, launched *after* their all new Z8000 "16 bitter" which is not itself Z80 compatible this new entry is aimed at correcting the swing away from Zilog caused by that omission. The Z800 will execute Z80 code while offering extra goodies such as an on-chip multiply/divide unit and lots of extra 16 bit facilities and Zilog hope that it will woo Z80 users into the Zilog 16 bit fold!

Also interesting is the NSC 800 from National which will be covered later in this series. The NSC 800 is a new CMOS low power device which executes Z80 code on the *inside* and yet looks like an 8085 (with a multiplexed bus) on the *outside*.

REGISTERS

The Z80 has the most complete set of registers of any 8 bit microprocessor and an instruction set which makes very good use of them. All the traditional 8080A registers are available, in fact there are *two* complete sets of data registers AF, BC, DE and HL and two instructions EX and EXX which allow the programmer to switch between the two banks. EX switches between AF and AF' and EXX switches between BC, DE, HL, and BC', DE', HL'. This facility is especially useful during interrupt routines since it allows the registers used by the main program to be saved by simply switching to the alternate bank, avoiding the usual chore of pushing the registers onto the stack to prevent their contents being destroyed by the interrupt routine. Very rapid response to interrupts is therefore possible, and this is important in real-time controller applications.

The Z80 has a single 16 bit Program Counter (PC) and a single 16 bit Stack Pointer (SP) and these are identical to their 8080 counterparts in all respects. There are however, four important new registers in the Z80 set which cannot be accessed by traditional 8080 instructions, and these are responsible for some of the more advanced features of Z80 operation.

You may remember that I criticised the 8080A/8085A for having no Index registers, and the 6800/6802 for having only one. Well the Z80 puts this right by having two full 16 bit Index registers IX and IY, making all the data processing fans deliriously happy, and making the task of compiler and interpreter writers much easier. Like the 6800 Index Register, the Z80 IX and IY are used as memory pointers with the special feature that the Register Indirect instructions which utilise them have a facility for specifying a one byte offset value providing easy access to tables of data stored in memory.

The two remaining Z80 registers have special significance and are not generally found in other microprocessors. The I register forms an important part of the special Z80 Mode 3 interrupt mechanism which is described later, and is used to hold the eight most significant bits of the Pointer to the interrupt vector table. During an interrupt the interrupting device provides the lower bits which when combined with the I register form a unique memory address for that particular device. Stored at that address will be the interrupt vector, or in other words the start address of the related interrupt routine. Normally the I register is set up by the programmer as part of the initialisation routine.

The R register is basically a seven bit counter used as a refresh address for Dynamic RAM memory. With other microprocessors special external hardware is used to carry out DRAM refresh, but the Z80 does the job itself by generating a refresh control signal and sending the contents of the R register out on the lower half of the address bus while the CPU is busy decoding and executing a previously fetched instruction. This technique saves external hardware and does not interfere with normal processor activity since it only uses the bus when it would otherwise be idle.

INSTRUCTION SET

If you already have a working familiarity with the 8080A instruction set then you are half way to knowing the Z80. *Only* half way though, because there are two snags: The Z80 has 158 instructions to the 78 of the 8080A, and even those instructions which are shared have had their names changed at the mnemonic level.

Perhaps the most important change of title concerns the data movement instructions which have various names (MOV, LXI, STAX, SHLD, MVI, LDAX, LDA, STA) on the 8080A, but which all form part of the load group using the LD mnemonic on the Z80. With the Z80 it is the source and destination parameters given

Z80 REFERENCE FILE SHEET

GENERAL

The Z80 was developed by a splinter group of Intel designers who saw the opportunity of developing a 'Super 8080'. Despite having many powerful new instructions and using different mnemonics, the Z80 is compatible with the 8080A and 8085A at the object code level, allowing existing 8080 software such as the CP/M operating system to run without modification. The Z80 has been enormously popular for both controller and data processing applications thanks to the useful blend of 8080 compatibility and a set of powerful new features such as extra registers, automatic refresh for dynamic RAMs, 5V single chip operation, and an enhanced instruction set.

REGISTERS

The Z80 has two banks of 8 general purpose 8bit registers although only one bank may be accessed without a bank switch instruction. In addition to the usual stack pointer and program counter the Z80 has two 16 bit index registers and the dedicated I and R regs.

A	F	A´	F´
B	C	B´	C´
D	E	D´	E´
H	L	H´	L´
INTERRUPT VECTOR I	MEMORY REFRESH R	ALTERNATE REGISTER SET	

INDEX REGISTER IX
INDEX REGISTER IY
STACK POINTER SP
PROGRAM COUNTER PC

NOTES:
1) EXX instruction used to switch between banks. Especially useful for fast interrupt context switch.
2) I register points to interrupt vector page in main memory
3) R register is counter for 16K DRAM refresh.

FLAGS:

D7	D6	D5	D4	D3	D2	D1	D0
SIGN	ZERO		AUX CARRY		PARITY & OVERFLOW	SUBTRACT	CARRY

INSTRUCTION SET AND SOFTWARE

Zilog have taken advantage of the 12 unused opcodes in the 8080A instruction set to add many new instructions to the Z80 set. Some of these spare codes are used directly, others are used as windows to extra opcode tables each of which provide 256 extra opcode possibilities. This technique makes it necessary to use two opcode bytes instead of one and results in the use of one, two, three and four byte instructions whereas most 8 bit devices have only one, two, or three byte types. The Z80 has 158 different instruction types, compared with 78 for the 8080A and has many new capabilities, such as relative jumps, bit set, reset and test, nibble shifts and block transfer and search. Because of its ancestry the Z80 instruction set is rather messy and more difficult to learn than that of say, the 6809 its main rival. It will execute 8080 object code and therefore has access to the CP/M operating system.

PERFORMANCE DATA

	Z80
MEMORY ADDRESS RANGE	64K
I/O ADDRESS RANGE	256
CLOCK FREQUENCY*	2.5MHz
POWER SUPPLIES	+5V
INTERRUPTS (3 MODES)	INT, NMI

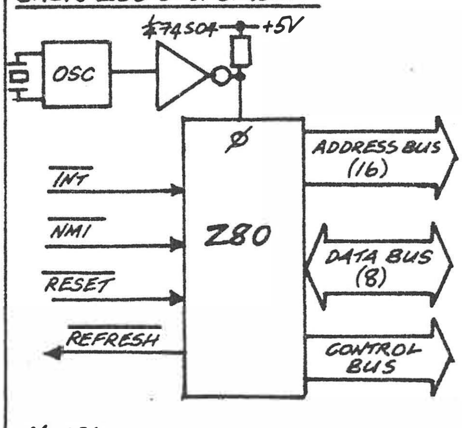
*NOTE: 4MHz VERSION (Z80A) ALSO AVAILABLE

BENCHMARKS	Z80
ADD REGISTER TO ACCUM	1.6µs
OUTPUT ACCUM TO PORT	4.8µs
MOVE FROM MEMORY TO MEMORY	6.4µs

Z80

Signal	Pin	Pin	Signal
A11	1	40	A10
A12	2	39	A9
A13	3	38	A8
A14	4	37	A7
A15	5	36	A6
Ø	6	35	A5
D4	7	34	A4
D3	8	33	A3
D5	9	32	A2
D6	10	31	A1
+5V	11	30	A0
D2	12	29	GND
D7	13	28	RFSH
D0	14	27	M1
D1	15	26	RESET
INT	16	25	BUSRQ
NMI	17	24	WAIT
HALT	18	23	BUSAK
MREQ	19	22	WR
IORQ	20	21	RD

BASIC Z80 CPU CIRCUIT



NOTE: Clock circuit simplified for clarity.

MANUFACTURERS

ORIGINATOR — ZILOG

2ND. SOURCE — MOSTEK, NEC, SHARP, SGS.

SUPPORT CHIPS

Z80 needs an external clock generator (Shottky T.T.L) and has a family of powerful peripheral chips with built-in interrupt controllers as follows:- Z80PIO (parallel ports), Z80CTC (timers), Z80DMA (direct memory access controller) and Z80SIO (dual USART)

after the LD mnemonic which specify just *what* is to be moved to *where,* a feature which makes life easier for the programmer but more difficult for those who have to write assemblers! The LD group has also been greatly extended by new instructions which provide access to the additional registers and new operations which use the existing 8080A style registers.

Apart from the LD group, there are several other groups which provide powerful new facilities which 8080A users could only dream of. Some of these new instructions plug directly into vacant spaces in the 8080A opcode byte table which of course provides 256 possibilities of which the 8080A uses only 242. Eight of these unused positions are filled with the six new relative jump instructions plus the two register bank exchange instructions already mentioned, but the remaining four opcodes are used to gain access to *all* the other new Z80 instructions by acting as "trap-door" exits to four additional 256 entry opcode tables. This is the only way in which an 8 bit processor can increase the range of opcodes available but of course it does mean that all the instructions in these new tables need a two byte, rather than a single byte, opcode.

The trapdoor provided by the CB(Hex) opcode is used to gain access to a table giving 248 new instructions which provide individual bit set, test, and reset, functions and an extended set of shift and rotate instructions which confers the ability to shift or rotate data in any general purpose register or any memory location.

The DDH and FDH trapdoor opcodes provide access to tables containing the many new instructions which act on, or use, the two Index registers IX and IY, and the EDH opcode invokes a table which accommodates all the miscellaneous instructions which do not logically belong in the other three. Whereas the first three tables mentioned contained variations on a simple theme, the table prefixed by the EDH opcode byte contains an interesting assortment of new features which would, by themselves, considerably boost the power of the 8080A had Intel seen fit to include them! In this table we find, for example, instructions to extend the 16 bit register arithmetic capability, to provide control of the enhanced Z80 interrupt technique, to allow the right and left rotation of BCD numbers in memory with the accumulator, and the powerful new feature of macro instructions which will operate repetitively on whole *blocks* of data.

To go with the extended instruction set of the Z80 there is an extended set of addressing modes, with Zilog claiming ten to Intel's four. All is not what it appears, however, since most manufacturers seem to have their own ideas (and names!) for what constitutes an addressing mode, with seven of the ten claimed by Zilog actually available on the 8080A anyway! The three real additions, are, however, very useful.

Zilog have corrected the omission by Intel of the Relative addressing mode. You may remember from the 6800 (which does have it) that this mode is used exclusively with jump instructions to move backwards and forwards in memory *relative* to the current program counter value, as distinct from direct or absolute jump instructions which must be provided with the address of a specific destination location in memory. The main advantage of the relative jump is that it uses only two bytes (Opcode + displacement) as opposed to the three bytes (Opcode + 2 byte address) required for an absolute jump, although it does also make possible the creation of small program segments which will execute *wherever* they are placed in memory. Like the 6800 the Z80 has a relative jump range of +127 to −129 locations.

Also available, thanks to the new index registers IX and IY, is Indexed Addressing which uses the registers as pointers to which a single byte offset is added to form an address. Once again this facility corrects an 8080A defect not shared by the 6800.

Finally, the Z80 offers the Bit Addressing mode, which, in combination with one of the other addressing modes, is used with the bit set, test, and reset instructions to act on a particular bit in any general purpose register or memory location. This facility is particularly useful in controller applications where a single byte can be used to provide eight separate "flag" bits to indicate status.

SOFTWARE

The Z80 has become extremely popular and is used in many well known microcomputers. Notable among these are the NASCOM, The ZX80 and 81, the TRS80, the Video Genie, the Sharp MZ80K, and the Superbrain. Since the Z80 uses a super-set of the 8080A opcodes it can utilise nearly all existing 8080A code including of course, the ubiquitous CP/M disc operating system. Despite the attraction of access to the vast range of CP/M compatible software there is the annoying limitation that CP/M does not take full advantage of the extended Z80 instruction set and this has prompted many microcomputer manufacturers to develop their own disc operating systems using all the new Z80 features to increase the speed and capability of their systems. Zilog themselves have a powerful DOS called RIO which is used on their own development systems, and the popular TRS80 has, in addition to the Tandy TRSDOS, other operating systems such as NEWDOS, NEWDOS +, and LDOS, all of which vie with each other to provide bigger and better features. For "home-brew" systems without discs there have been a number of books published which include standalone software in the form of Monitors, Editors and Assemblers. One which I can recommend is "Practical Microcomputer Programming: The Z80" by W. J. Weller.

INTERFACING

As you may expect with a powerful chip like the Z80 there is quite a lot to learn about interfacing it to the outside world, more, in fact, than I can hope to cover in detail here. Those planning a "homebrew" system should obtain one of the many useful books written about the Z80 which provide more detail than the rather skimpy Zilog technical manual.

Anyone contemplating the direct substitution of a Z80 for an 8080A to increase performance will be rather put off at first by the pin-out differences between the two chips. Not only are the pin assignments themselves quite different, but so too are the names and functions of several of the control lines. For this reason it is necessary to carefully consider and understand the relationships between the two processors before a substitution is attempted. For example, the 8080A in combination with the necessary 8228 system controller chip generates four lines to control bus transactions, namely $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$ and $\overline{I/OW}$ the functions of which are self explanatory. The Z80 also uses four lines, but in its case the names are $\overline{RD}$, $\overline{WR}$, $\overline{IOREQ}$, and $\overline{MREQ}$, which are also self explanatory but *different* to those of the 8080A. Once these differences are appreciated it is quite easy to generate the 8080A equivalents by simple gating. The 8080A $\overline{MEMR}$ can be obtained by ANDing together $\overline{RD}$ and $\overline{MREQ}$ and the other signals can be generated in a similar fashion.

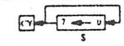
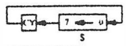
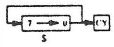
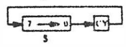
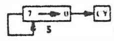
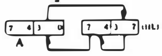
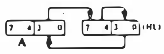
One aspect of the Z80 which needs careful attention is the clock input Ø. In my original data sheet Zilog make the naughty claim that this single phase TTL level clock "requires only a 330 ohm pull-up resistor to +5 Volts to meet all clock requirements." Unfortunately, this simple expedient of adding a pull-up resistor to a TTL clock oscillator cannot be guaranteed at the higher clock frequencies. What you have to do in practice is to use a Shottkey TTL driver such as 1/6 of a 74S04 to which you add a *pnp* transistor, three resistors and a capacitor. This may explain why many Z80 users (e.g. Tandy with their TRS80) do not run their processors at the data-sheet maximum of 2.5MHz. It also dilutes somewhat the Zilog claim that the Z80 provided the first one-chip processor group, since a 74S package and several discretes must take up at least as much room as the special clock generators used by the 8080A and the 6800!

Another couple of problems which were not at first acknowledged by the manufacturers concern the operation of the processor with dynamic RAM. If the contents of DRAM memory must survive a Z80 RESET, then a hardware "fix" consisting of several TTL packages and assorted discretes must be added. Also, correct operation with DRAMs cannot be guaranteed anyway unless the upper four bits of the Z80 address bus are latched externally in a 74LS75 or similar register to prevent random data loss. It seems that there is no escape from the second problem, even though it may not occur in all systems, but the best approach to the first problem is to avoid the use of RESET except at power-on time and to use the NMI interrupt for all warm-starts when memory data must be conserved. Don't let these DRAM problems put you off the Z80 too much though, since the remedies are fairly painless, and in all other respects the Z80 is an ideal processor for DRAM systems thanks to its on-board refresh system.

One of the neatest innovations provided by the Z80 is a really powerful and versatile interrupt structure which has three separate

INSTRUCTION SET SUMMARY

Group	Mnemonic	Symbolic Operation	Comments
8-BIT LOADS	LD r, s	r ← s	s ≡ r, n, (HL), (IX+e), (IY+e)
	LD d, r	d ← r	d ≡ (HL), r (IX+e), (IY+e)
	LD d, n	d ← n	d ≡ (HL), (IX+e), (IY+e)
	LD A, s	A ← s	s ≡ (BC), (DE), (nn), I, R
	LD d, A	d ← A	d ≡ (BC), (DE), (nn), I, R
16-BIT LOADS	LD dd, nn	dd ← nn	dd ≡ BC, DE, HL, SP, IX, IY
	LD dd, (nn)	dd ← (nn)	dd ≡ BC, DE, HL, SP, IX, IY
	LD (nn), ss	(nn) ← ss	ss ≡ BC, DE, HL, SP, IX, IY
	LD SP, ss	SP ← ss	ss ≡ HL, IX, IY
	PUSH ss	(SP-1) ← ss_H; (SP-2) ← ss_L	ss ≡ BC, DE, HL, AF, IX, IY
	POP dd	dd_L ← (SP); dd_H ← (SP+1)	dd ≡ BC, DE, HL, AF, IX, IY
EXCHANGES	EX DE, HL	DE ↔ HL	
	EX-AF, AF′	AF ↔ AF′	
	EXX	$\begin{pmatrix} BC \\ DE \\ HL \end{pmatrix} \leftrightarrow \begin{pmatrix} BC' \\ DE' \\ HL' \end{pmatrix}$	
	EX (SP), ss	(SP) ↔ ss_L, (SP+1) ↔ ss_H	ss ≡ HL, IX, IY
MEMORY BLOCK MOVES	LD1	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC–1	
	LDIR	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC–1 Repeat until BC = 0	
	LDD	(DE) ← (HL), DE ← DE–1 HL ← HL–1, BC ← BC–1	
	LDDR	(DE) ← (HL), DE ← DE–1 HL ← HL–1, BC ← BC–1 Repeat until BC = 0	
MEMORY BLOCK SEARCHES	CP1	A–(HL), HL ← HL+1 BC ← BC–1	
	CPIR	A–(HL), HL ← HL+1 BC ← BC–1, Repeat until BC = 0 or A = (HL)	A–(HL) sets the flags only. A is not affected
	CPD	A–(HL), HL ← HL–1 BC ← BC–1	
	CPDR	A–(HL), HL ← HL–1 BC ← BC–1, Repeat until BC=0 or A=(HL)	
8-BIT ALU	ADD s	A ← A + s	
	ADC s	A ← A + s + CY	CY is the carry flag
	SUB s	A ← A – s	
	SBC s	A ← A – s – CY	s ≡ r, n, (HL) (IX+e), (IY+e)
	AND s	A ← A ∧ s	
	OR s	A ← A ∨ s	
	XOR s	A ← A ⊕ s	
	CP s	A – s	s = r, n (HL) (IX+e), (IY+e)
	INC d	d ← d + 1	d = r, (HL) (IX+e), (IY+e)
	DEC d	d ← d – 1	
	ADD HL, ss	HL ← HL + ss	ss ≡ BC, DE HL, SP
	ADC HL, ss	HL ← HL + ss + CY	
	SBC HL, ss	HL ← HL – ss – CY	
	ADD IX, ss	IX ← IX + ss	ss ≡ BC, DE, IX, SP

Group	Mnemonic	Symbolic Operation	Comments
16-BIT ARITHMETIC	ADD IY, ss	IY ← IY + ss	ss ≡ BC, DE, IY, SP
	INC dd	dd ← dd + 1	dd ≡ BC, DE, HL, SP, IX, IY
	DEC dd	dd ← dd – 1	dd ≡ BC, DE, HL, SP, IX, IY
GP ACC. & FLAG	DAA	Converts A contents into packed BCD following add or subtract.	Operands must be in packed BCD format
	CPL	A ← $\overline{A}$	
	NEG	A ← 00 – A	
	CCF	CY ← $\overline{CY}$	
	SCF	CY ← 1	
MISCELLANEOUS	NOP	No operation	
	HALT	Halt CPU	
	DI	Disable Interrupts	
	EI	Enable Interrupts	
	IM 0	Set interrupt mode 0	8080A mode
	IM 1	Set interrupt mode 1	Call to 0038_H
	IM 2	Set interrupt mode 2	Indirect Call
ROTATES AND SHIFTS	RLC s		
	RL s		
	RRC s		
	RR s		
	SLA s		s ≡ r, (HL) (IX+e), (IY+e)
	SRA s		
	SRL s		
	RLD		
	RRD		
BIT S, R, & T	BIT b, s	Z ← $\overline{s_b}$	Z is zero flag
	SET b, s	s_b ← 1	s ≡ r, (HL) (IX+e), (IY+e)
	RES b, s	s_b ← 0	
INPUT AND OUTPUT	IN A, (n)	A ← (n)	
	IN r, (C)	r ← (C)	Set flags
	INI	(HL) ← (C), HL ← HL + 1 B ← B – 1	
	INIR	(HL) ← (C), HL ← HL + 1 B ← B – 1 Repeat until B = 0	
	IND	(HL) ← (C), HL ← HL – 1 B ← B – 1	
	INDR	(HL) ← (C), HL ← HL – 1 B ← B – 1 Repeat until B = 0	
	OUT(n), A	(n) ← A	
	OUT(C), r	(C) ← r	
	OUTI	(C) ← (HL), HL ← HL + 1 B ← B – 1	
	OTIR	(C) ← (HL), HL ← HL + 1 B ← B – 1 Repeat until B = 0	
	OUTD	(C) ← (HL), HL ← HL – 1 B ← B – 1	
	OTDR	(C) ← (HL), HL ← HL – 1 B ← B – 1 Repeat until B = 0	

	Instruction	Operation	Conditions
JUMPS	JP nn	PC ← nn	cc: NZ PO, Z PE, NC P, C M
	JP cc, nn	If condition cc is true PC ← nn, else continue	
	JR e	PC ← PC + e	
	JR kk, e	If condition kk is true PC ← PC + e, else continue	kk: NZ NC, Z C
	JP (ss)	PC ← ss	ss = HL, IX, IY
	DJNZ e	B ← B − 1, if B = 0 continue, else PC ← PC + e	
CALLS	CALL nn	(SP−1) ← PC_H (SP−2) ← PC_L, PC ← nn	cc: NZ PO, Z PE, NC P, C M
	CALL cc, nn	If condition cc is false continue, else same as CALL nn	
RESTARTS	RST L	(SP−1) ← PC_H (SP−2) ← PC_L, PC_H ← 0 PC_L ← L	
RETURNS	RET	PC_L ← (SP), PC_H ← (SP+1)	
	RET cc	If condition cc is false continue, else same as RET	cc: NZ PO, Z PE, NC P, C M
	RETI	Return from interrupt, same as RET	
	RETN	Return from non- maskable interrupt	

In the table the following abbreviations are used.

b	≡	a bit number in any 8-bit register or memory location
cc	≡	flag condition code
NZ	≡	non zero
Z	≡	zero
NC	≡	non carry
C	≡	carry
PO	≡	Parity odd or no over flow
PE	≡	Parity even or over flow
P	≡	Positive
M	≡	Negative (minus)
d	≡	any 8-bit destination register or memory location
dd	≡	any 16-bit destination register or memory location
e	≡	8-bit signed 2's complement displacement used in relative jumps and indexed addressing
L	≡	8 special call locations in page zero. In decimal notation these are 0, 8, 16, 24, 32, 40, 48 and 56
n	≡	any 8-bit binary number
nn	≡	any 16-bit binary number
r	≡	any 8-bit general purpose register (A, B, C, D, E, H, or L)
s	≡	any 8-bit source register or memory location
s_b	≡	a bit in a specific 8-bit register or memory location
ss	≡	any 16-bit source register or memory location

subscript "L" ≡ the low order 8 bits of a 16-bit register
subscript "H" ≡ the high order 8 bits of a 16-bit register
( ) ≡ the contents within the ( ) are to be used as a pointer to a memory location or I/O port number
8-bit registers are A, B, C, D, E, H, L, I and R
16-bit register pairs are AF, BC, DE and HL
16-bit registers are SP, PC, IX and IY

modes of operation, one of which must be selected by the programmer during initialisation.

During system reset the processor is forced into Interrupt Mode 0, which is identical to the interrupt scheme of the 8080A in which external hardware provides a single byte Restart or a three byte Call instruction following interrupt acknowledge. This mode is ideal for use with an external interrupt controller chip such as the Intel 8259. The other modes are selected by issuing an IM 1 or IM 2.

Mode 1 is a scheme for simple systems which have only one interrupt source, and consequently needs no external interrupt controller. When in this mode, an interrupt on the Z80 INT line causes an immediate restart to location 0038H where the interrupt routine itself may be positioned, or, more likely, a Jump instruction to a routine elsewhere in memory.

Mode 3 is the most powerful interrupt response mode since it provides fast vectoring to up to 128 separate interrupt routines which may be located anywhere in memory. A practical restriction to this mode is that it is really only useful for systems using Zilog peripheral chips which all have the necessary control logic built in. Using this mode, when an interrupt is accepted the interrupting device supplies a single byte which is combined with the contents of the I register to give a 16 bit address which is used as a pointer to a unique entry in a table containing up to 128 possible interrupt vectors. This table can be located on any page boundary, as determined by the programmer when he loads the I register. Both the I register and the contents of the vector table must be set up during the initialisation routine.

In Mode 3 the 128 possible interrupting "devices" each get a very rapid response from the Z80, but what if one device is already being serviced when another generates an interrupt of its own? This problem is solved by the so-called daisy-chain priority scheme which is built into all Zilog peripheral devices along with registers to supply the single byte needed to form the vector table address. There are only four basic peripheral devices in the Zilog family, but these cater for most requirements, as follows:

The Z80 PIO provides two parallel 8 bit I/O ports with "handshake" lines and each port may be separately programmed into the byte output, byte input, or individual bit I/O mode with Port A also offering a bidirectional "bus" mode.

The Z80 CTC provides four independent channels for counting and timing applications with each channel having an 8 bit prescaler which can be set to divide by 16 or 256, and an 8 bit down counter which counts down to zero after being preset from an 8 bit time constant register loaded by the programmer.

The Z80 DMA chip can take control of the system bus to transfer data to and from memory directly without Z80 processor participation. A wide variety of DMA transfers are programmable with data moved a byte at a time or in blocks, and with address registers and block counters automatically incrementing or decrementing.

The Z80 SIO provides the USART function for the Z80 but unlike the Intel 8251 the SIO is a dual channel device containing two complete serial I/O channels capable of operating at up to 550K bits per second in synchronous or asynchronous mode.

All four of the Z80 peripherals are fully compatible with Mode 3 interrupt operation and utilise a unique daisy-chain priority scheme. Each device has an Interrupt Enable In (IEI) and Interrupt Enable Out (IEO) in addition to its conventional interrupt output. While the interrupt outputs from several devices are connected in parallel to the Z80 INT pin, the IEI/IEO lines of the various devices are connected in series to form the daisy-chain. The devices electrically "closer" to the CPU in the chain have a higher priority than those "further away", since when a device requests an interrupt, it no longer gives an IEO output to devices further down the chain. Only the device with its IEI line high (= enabled) and its IEO line low (= requesting) can respond to the interrupt acknowledge signal from the processor by sending its pre-programmed interrupt vector byte on to the bus. Devices already being serviced can be interrupted themselves, but only by devices with a higher priority.

The special interrupt capability built into the Z80 peripherals can be used to advantage in many systems, although it does have its penalties since the complex Z80 devices are generally more expensive than the simpler 8080A and 6800 family chips, and many would argue that the SIO is much *too* clever for most applications.

APPLICATIONS

There can be no doubt that the Z80 is one of the most powerful 8 bit devices ever developed with sufficient flexibility to be used in both controller and data processing applications, and with a very broad user software base. These convincing advantages must be weighed very carefully against its rather less desirable features like the confusing instruction set and the hardware problems, before any potential user takes the plunge. I certainly feel that those without previous 8080A experience or those who wish to learn about microprocessors from scratch should take a careful look at the 6809 or even the 6502 before deciding. Readers who already have a Z80 tucked away in a BASIC orientated home computer can take comfort in the fact that they can take full advantage of Z80 power *without* worrying about the problems.

Teach yourself Assembler

Starting this month we begin APC's *introduction to assembly language programming. In order to give you the best possible understanding of it, Paul Overaa uses Basic as his reference point.*

If you've been wondering whether you would ever learn to program in assembly language . . . if you've picked up a teach yourself book and been blinded by science . . . if you've wished you could join the other *APC* readers for whom Subset is a monthly treasure trove . . . then today is your red-letter day. Each article will deal with a digestible section of assembler and will contain a piece of code for you to type in and run: to do this you will need an assembler program. You can buy one for your computer from a software dealer. Our series will contain details for computers using the Intel 8080, the Zilog Z80 or the MOS 6502 processors, but if your machine doesn't happen to be based on one of these, don't despair. Throughout the series emphasis will be placed on the programming structures used, so you should be able to implement the example programs using the instruction set appropriate to your computer's processor.

What is an assembly language?

Your first question may be . . . What is an assembly language anyway? The language a microprocessor understands — machine language — is that of binary numbers. For example, the binary number 11000011, (a jump instruction in the language of one processor), is rather like a GOTO in Basic. The difference is that a memory location instead of a Basic line-number is targeted. It's possible to program a computer using these codes, or using hexadecimal numbers in their place, but the moment you use any system other than binary, the computer cannot recognise the instruction. Instructions need to be converted into binary, either by hand or using the computer itself, before the processor can act on them.

Hexadecimal numbers are almost as incomprehensible to humans as are binary ones: would it matter to you whether you had to use 11000011 or C3 instead of a word like GOTO? There is certainly a difference, and early on in the development of the computer it became clear that to write even moderately sized programs in binary, hexadecimal ('hex') or octal was a masochistic activity. So a language was created that used operations identical to those the computer could perform, but which was easier for humans to understand. That's how assembly language programming was born. The names given to the instructions are mnemonics (a word meaning 'memory jogger'). The mnemonic for the 11000011 instruction is JMP, which is a useful improvement.

Hexadecimal numbers

You'll still need to manipulate some hexadecimal numbers as memory locations. Don't panic! You've probably already used hexadecimal counting unconsciously. When you'd add pounds and ounces, you'd add the ounces first: if they came to more than 16 you'd carry the number of multiples of 16 into the pounds column, for example:

2lbs 14 ounces
+
2lbs 5 ounces

5lbs 3 ounces
(14+5=19 which is 'one of 16' and '3 oz left over')

If, instead of using numbers from 0 to 15 for ounces we used 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, we could then write 2lbs 14oz as 2E and so on. This extended numbering system forms the basis of the hexadecimal notation. If you now consider the eight bits of a byte of binary information as two groups of four bits, you will appreciate that we can represent each of those groups by a hexadecimal digit. So, with binary number 10001111.

1000 Binary=8 Decimal=8 Hex
1111 Binary=15 Decimal=F Hex,

so 10001111 binary can be written very compactly as 8F hex. An eight bit binary number can take values from 0 to 255, that is, from 00000000 binary to 11111111 binary. Only two characters are needed to express the same range in hexadecimal notation! In a similar fashion, we can represent two bytes (16 bits) of information by using four hex digits: try writing 1111000010001111 in hex yourself. Most assemblers expect memory locations to be provided as hexadecimal numbers.

Standard mnemonics

The choice of mnemonics to be used with a particular processor is arbitrary, and the selection is made and recommended by the people who make the chip in question. If you think in terms of Basic, you may have realised that different software houses write their Basic interpreters and compilers around various standard facilities, but still end up producing slightly different versions of Basic. If you compare the Basic keywords for the Apple with those for the VIC-20 you'll see the difference. That's why you need the *APC* converter chart when you want that wizard ZX81 program of Uncle Sam's to run on your Tandy. Manufacturers of microprocessors also design differently, so that each type has its own characteristics and methods of implementing the functions it provides, thus its own set of mnemonics. For this reason, the mnemonics you use will depend on the chip around which your computer is built.

In this series we deal with three chips, the Intel 8080, the Zilog Z80 and the MOS 6502. The Z80 is used, for example, in the System 80 and MicroBee; the 6502 in the Apple and Pet, to name only a couple. The 8080 is included because although getting on in years, it is a useful jumping-off point for some of our explanations.

1 of 6.

Mar 84 5(3)

What is an assembler program?

Having written in assembly language mnemonics, you need to convert your program into binary for the computer to work. The computer program that does this conversion for you is called an assembler. Nowadays, many assembler programs have editing facilities incorporated. This means you can write your assembly language program direct into the computer as you would a Basic program, and the editor will pick you up on your errors. Then, when the program is bug-free, you press a key and the program is 'assembled'. The assembler itself can be on a cassette, like a games program, which you load into your machine.

Each assembler has its own rules, which you will need to be aware of. One of the things you will learn this month is how to use your assembler to get the code in this article into your machine. Have a glance through the documentation supplied with your assembler before you begin, but don't worry if a lot of it seems like incomprehensible jargon to start with.

A line in an assembly language program can be divided into three regions or 'fields'. These are a label field, an instruction field and a comments field. The first and last are optional. Your assembler will have fixed rules for identifying the individual fields and you'll find these in the documentation provided with it (see Fig 1, example line from an assembly language program).

CHECK^CHARACTER:	CPI CARRIAGE RETURN	: end of input?
Label field (Optional)	Instruction field	Comment field (Optional)

Fig 1 Example line from an assembly language program

Comments within the program

You'll probably be familiar with REM statements in Basic. In such a high-level language you can often work out what a program does even if it hasn't been properly documented with a healthy sprinkling of REM statements. The same is not true of assembly language. One of the fundamental differences between high level languages (such as Basic) and assembly language is that the latter is difficult to analyse. This is because any inherent structure is not always obvious in the code, so make the most of a lesson many of us have learnt the hard way and document to the maximum extent, time and your assembler permitting. *The importance of placing understandable comments within an assembly language program cannot be over-emphasised.*

Labels

Most assemblers allow you to use meaningful names for specific locations in memory. This means you don't have to remember, for example, that location 0A3F Hex is the start of a subroutine that checks for keyboard input characters. Instead, you may use a label, say CHECK^CHARACTER in the label field of the first instruction of that subroutine. The assembler will add the label to an internal symbol table it maintains, and you can then use the label to reference the routine. (Some assemblers require you to place a delimiter character, often a colon, immediately after a label definition in order that the assembler can distinguish it from the instruction field.)

Equate directive

Another facility offered by assemblers enables names to be assigned to numeric values. Since this facility is a function of the assembler rather than the processor, the equate directive is known as a pseudo operation, pseudo-op for short. It is especially useful for defining many of the common ASCII characters.

By placing the following statement at the start of an assembly language program, you cause the assembler to include these definitions in its symbol table.

```
CARRIAGE$RETURN        EQU 13
SPACE                  EQU 32
```

Using these definitions will make your programs more readable to yourself . . . and to others.

Other pseudo-ops such as ORG (short for origin) are available: ORG selects whereabouts in memory your program is to start. Having given you an idea of what assembly language is, we'll add to your knowledge of pseudo-ops as and when required.

Operating systems

Before a microprocessor can do anything useful, it needs a means of getting data, somewhere to send its output, additional memory, and a way of coordinating everything that's going on. The routines which perform these and other functions are known collectively as the computer's operating system. Micros that use disk drives for input/output data generally use one of a number of 'soft' operating systems such as CP/M (Control Program for Microcomputers) or MS-DOS (Microsoft Disk Operating System). These are programs which the user needs to load into his machine before any other work is done. Home micros generally have an 'own-brand' operating system resident in read only memory (ROM).

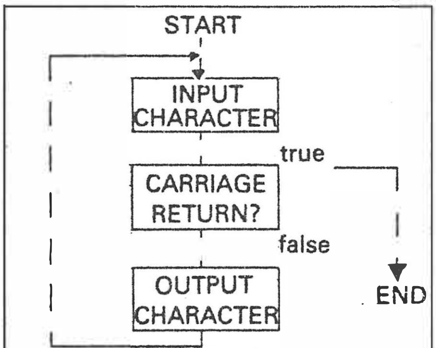


Fig 2 Flowchart for the example program

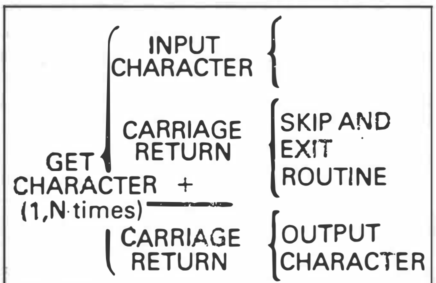


Fig 3 Warnier diagram for the example program

Operating systems usually contain accessible routines which you may use in your own programs to simplify many operations. In this series we can't cover *all* the available operating systems, so we'll avoid reference to specific computers. This means that when the time comes, you will have to delve into your computer's manual for certain details you'll need. But by the time we get to that stage you'll have been given sufficient grounding in the general principles to know what you're looking for.

Memory pages

I've already said that it's possible with a two byte address to specify any one memory location out of a total of 64k (64 × 1024=65536) such locations. Such an address can be written in hex form

2 of 6. Mar 84. 5(3)

using four digits. An additional concept of dividing available memory into pages, each of 256 locations, has proved useful. The first byte, or high byte, of such an address is often called the page number. Page zero refers to the initial 256 bytes of memory, whose addresses go from 0000 hex to 00FF hex. Structured programming has been made possible thanks to the discovery that virtually all problems can be solved using a combination of three structures, sequence, repetition and alternation.

For our theme this month, let's take repetition — the loop structure. The program we'll use to illustrate it, and to give you your first hands on

```
5  ***************************** REM  SETUP BLOCK  **********
10 CLEAR
20 CARRIAGE.RETURN$=CHR$(13)
22 ***************************** REM  END SETUP BLOCK  *******
23
24
25 ***************************** REM  MAIN BLOCK *************
30 X$=INPUT$(1)                  REM  COLLECT CHARACTER IN X$
40 IF X$=CARRIAGE.RETURN$
   THEN 70                       REM  END OF INPUT IF TRUE
50 PRINT X$                      REM  OUTPUT CHARACTER TO VDU
60 GOTO 30                       REM  GET NEXT CHARACTER
62 ***************************** REM  END MAIN BLOCK  *********
63
64
65 ***************************** REM  END BLOCK
70 END
72 ***************************** REM  END OF END BLOCK
```

Fig 5 Complete Basic Version

experience of assembly language programming, is a simple one to collect characters directly from the keyboard and print them on the VDU screen. This program could be written in various ways in Basic. We've chosen a representation that makes for easy comparison with the assembly language equivalent. Figs 2 and 3 contain the flowchart and Warnier diagram for our program.

In use

The Basic program can be divided into three parts. An initial or 'set up' block is used to define a variable called CARRIAGE.RETURN$ as the string equivalent of the ASCII 13 carriage return code. The end block is nothing more than a single Basic END statement. The bulk of the program, labelled the main block, performs several functions. The INPUT$() function collects a character. We then check to see if it's a carriage return character, and if it is we jump to the end of the program. Otherwise, we print the character and jump back to the input statement in line 40 for the next character. An example of the basic form can be seen in Fig 5.

When we write an equivalent program in assembly language we use the same type of program structure. The set up block will, however, vary according to your processor, assembler and operating system. We give three examples of the kind of coding that could be expected. First we shall deal with the 8080 form:

8080 MNEMONICS FOR THE SET UP BLOCK

```
CARRIAGE$RETURN     EQU   13
                    ORG   100H
                    JMP   STACK
                    ORG   150H
STACK:              LXI   S,P,$-2
```

Operations used

1. Define CARRIAGE$RETURN so that the assembler will recognise this term as meaning the number 13.
2. Define whereabouts in memory your program is to start. Our assembler uses ORG, so in the example above the program starts at 100 hex.
3. We perform an unconditional jump to an address labelled STACK. A stack is an area of memory which is used to store items the processor will need from time to time. It's called a stack because items are added to it and taken from it in the same way you would take and add cards to and from the top of a pack of playing cards on the last in, first out basis. Each of the three processors we are describing has a stack pointer register to determine the memory location currently at the top of the stack. Instructions for placing items on top of the stack automatically adjust the stack pointer accordingly. We'll be using the stack often, and will deal with its other uses as appropriate. It is common practice to talk of the placing of data on the stack as 'pushing' onto the stack. When items are removed the terms popping or pulling are used.

A register is a place within a processor that can hold binary information. With eight bit processors we talk of an eight bit word length, and this, as you probably know, is called a 'byte'. 8 bit registers in our processors can therefore hold one byte of information.

Eight bit processors usually provide some means of combining pairs of their eight bit registers in order that a 16 bit (two byte) memory address can be specified. This is vital if we are to be able to 'address by name' all the memory locations in the 0 to 64k range. In fact, certain memory addressing instructions enable less than the full address to be given and this can have advantages in terms of speed of operation.

Since we've digressed, lets complete our digression by looking at a schematic layout of the 8080 processor (Fig 4). The 'Accumulator' is a straightforward 8 bit register, and is used, often

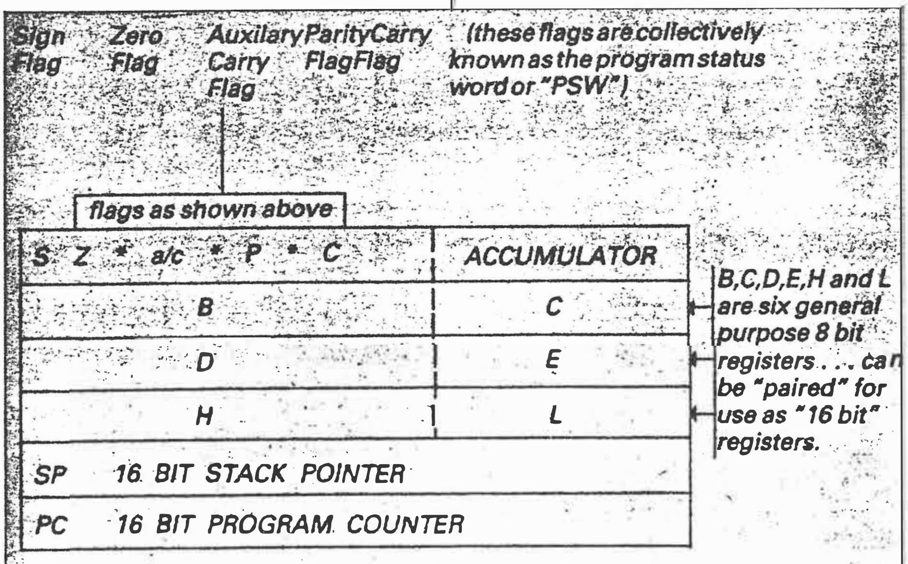


Fig 4 Schematic layout of the 8080 microprocessor

implicitly, for all data I/O and arithmetic/ Boolean operations. Many instructions apply specifically to this register.

Six 'Secondary Registers' are shown. Each can be used as an individual 8 bit register, but it's possible to pair B,C...D,E..and H,L to create effective 16 bit registers.

You'll also notice a set of 'Flags'. Flags are a group of bits which collectively can be referred to as a status word or program status word. These bits are affected by the occurrence of certain conditions: for example, any arithmetic operation that results in zero being present in the accumulator will set the zero flag to 1. (Remember that a bit can only take the value of 0 or 1), and that by convention, 1 is used to represent the 'true' condition.

That's the end of the digression, so back to the code. The unconditional jump we performed (JMP STACK) is the first real assembly language instruction we have encountered. It's called an unconditional jump because it is performed irrespective of any processor flag conditions. The mnemonic JMP represents a three byte instruction. The first byte is the op code, that is, the numerical representation of the mnemonic. The second and third bytes are the address that is specified.

The microprocessor performs this jump by placing the address following the op code into the program counter register, which is the destination register for the information transfer. As the mnemonic JMP uses the two byte address (or a label) as its operand it is said to be able to use the immediate addressing mode.

4. Now we tell the assembler to pass over the space we are reserving for use as a stack. The ORG 150H instruction means the assembler places our next mnemonic at this new origin, leaving half a page of memory for the stack.

5. Lastly, we load the stack pointer register (SP) with the value $-2. This is because our assembler uses '$' to define the address of the current memory location, and is done with the mnemonic LXI, used by the 8080 to place a 16 bit address into a register pair, in this case SP.

Remember to check with your assembler manual whether the pseudo-ops ORG and EQU are achieved using these or different mnemonics.

Z80 MNEMONICS FOR THE SET UP BLOCK

The only difference is the unconditional jump mnemonic which on the Z80 is JP. We again use immediate addressing.

```
CARRIAGE$RETURN     EQU 13
                    ORG 100H
                    JP STACK
                    ORG 150H
STACK:              LD SP,$-2
```

6502 MNEMONICS FOR THE SET UP BLOCK

The 6502 uses page one addresses for the stack. The stack pointer is an eight bit register but using an extra leading bit (implied with a bit of hardware jiggery pokery) the 6502 creates a nine bit address for the stack pointer. So, if you load it with FF hex it will be pointing to memory location 1FF. You can't load the stack pointer register directly on the 6502, so instead you load the X register using mnemonic LDX and then transfer the contents of X to the stack pointer (S) register using TXS. With 6502 systems the stack will have been set up by the operating system, so you will use the existing stack. Typical code for the 6502 set-up block is shown below:

```
CARRIAGE$RETURN     EQU 13
                    OKG 800H
```

We have indicated the general type of set up block usually required. It may be that your particular system requires joint use of the stack and that your programs should simply use an existing stack. In other cases, it is necessary to save the 'operating systems' stack pointer so that it can be reinstated when your program has finished. You must, to a large extent, be

FULL LISTING 8080 VERSION

Notes: The operating system we are using requires that you identify the system function needed by placing a 'function number' into the microprocessors C register. It also expects 'output characters' to be in the E register and not the accumulator. This means we have to use instructions to transfer the contents of the accumulator into the E register. We set up the necessary details and then we CALL the operating system through a common entry point which is a jump located at memory location 5. The direct I/0 function used also needs FF hex present in the E register to indicate that imput (rather than output) is required.

SET-UP-BLOCK

```
CARRIAGE$RETURN      EQU   13
OPERATING$SYSTEM     EQU   5
                     ORG   100H
                     JMP   STACK
                     ORG   150H
STACK:               LXI   SP,$-2
```

MAIN BLOCK

```
START:               CALL  INPUT$ROUTINE
                     CPI   CARRIAGE$RETURN
                     JZ    FINISH
                     CALL  OUTPUT$ROUTINE
                     IMP   START
```

END-BLOCK

```
FINISH               JMP   0                   ;Reboot operating
                                               system
```

INPUT-ROUTINE

Notes: We have to use a 'wait for input' loop here. With our system it is necessary to preserve registers before using the operating system 'calls'.

```
INPUT$ROUTINE:       PUSH B!PUSH D! PUSH H     Preserve registers
INPUT$ROUTINES1:     MVI   E,0FFH              ;Signifies console
                                               input
                     MVI   C,6                 Direct console 1/0
                                               function
                     CALL  OPERATING$SYSTEM
                     CPI   0                   ;0= no key preseed
                     JZ    INPUT$ROUTINES1
                     POP H ! POP D ! POP B     ;Restore registers
                     RET
```

OUTPUT-ROUTINE

```
OUTPUT$ROUTINE:      PUSH PSW ! PUSH B ! PUSH D ! PUSH H
                     MOV   E,A                 ;transfer is in E reg-
                                               ister
                     MVI   C,2                 ;Console output
                                               function
                     CALL  OPERATING$SYTEM
                     POP H ! POP D ! POP B ! POP PSW
                     RET
```

4 of 6. Mar. 84 5 (3)

```
8080 MNEMONICS
START: CALL   INPUT$ROUTINE          :collect character in
                                     :accumulator
       CPI    CARRIAGE$RETURN        :end of input if true
       JZ     FINISH
       CALL   OUTPUT$ROUTINE         :output character to VDU
       JMP    START                  :get next character
Z80 MNEMONICS
START: CALL   INPUT$ROUTINE          :COLLECT CHARACTER in
                                     ;accumulator
       CP     CARRIAGE$RETURN        :end of input if true
       JP     Z, FINISH
       CALL   OUTPUT$ROUTINE         :output character to VDU
       JP     START                  :get next character
6502 MNEMONICS
START: JSR    INPUT$ROUTINE          :collect character in
                                     :accumulator
       CMP    # CARRIAGE$RETURN      :end of input if true
       BEQ    FINISH
       JSR    OUTPUT$ROUTINE         :output character to VDU
       JMP    START                  :get next character
```

Fig 6 Main block coding

FULL LISTING Z80 VERSION

Notes: See the 8080 notes for details concerning our operating system

```
                         SET-UP-BLOCK
CARRIAGE$RETURN     EQU   13
OPERATING$SYSTEM    EQU   5
                    ORG   100H
                    JP    STACK
                    ORG   150H
STACK:              LD    SP,$-2
                         MAIN-BLOCK
START:              CALL  INPUT$ROUTINE
                    CP    CARRIAGE$RETURN
                    JP    Z,FINISH
                    CALL  OUTPUT$ROUTINE
                    JP    START
                         END-BLOCK
FINISH              JP    0                  ;Reboot operating
                                             system
                         INPUT-ROUTINE
Notes: See the 8080 notes for details concerning our operating system
INPUT$ROUTINE:      PUSH B ! PUSH D ! PUSH H ;Preserve registers
INPUT$ROUTINES1:    LD    E,0FFH             ;Signifies console
                                             input
                    LD    C,6                ;Direct console 1/0
                                             function
                    CALL  OPERATING$SYSTEM
                    CP    0                  ;0= no key preseed
                    JP    Z,INPUT$ROUTINE$1
                    POP H ! POP D ! POP B    ;Restore registers
                    RET
                         OUTPUT-ROUTINE
OUTPUT$ROUTINE:     PUSH PSW ! PUSH B ! PUSH D ! PUSH H
                    LD    E,A                ;transfer is in E reg-
                                             ister
                    LD    C,2                ;Console output
                                             function
                    CALL  OPERATING$SYSTEM
                    POP H ! POP D ! POP B! POP PSW
                    RET
```

guided by your own system requirements.

As you can see in Fig 6, the structure of all three examples of main block coding is identical. We first call INPUT$ROUTINE to collect the character in the accumulator: note that the 6502 mnemonic for a subroutine call is different from the Z80 and the 8080. The address INPUT$ROUTINE is put into the program counter to achieve a jump to the required subroutine. Prior to this the processor's program counter, which points to the next instruction, is automatically pushed onto the stack. When the subroutine ends, this address is popped off the stack and replaced in the program counter, which then points to the instruction after the CALL or JSR instruction.

The next instructions compare the character collected in the accumulator with the value CARRIAGE$RETURN. The processor subtracts the contents of the accumulator from the value of the byte specified (in this case 13). The zero flag is set if the result is zero but the result of the subtraction is not stored anywhere, nor are the contents of the accumulator altered.

Immediately following the comparison test we have used a conditional jump instruction. If the zero flag has been set, a jump to an as yet unspecified FINISH routine will follow.

If the zero flag has *not* been set, instead of jumping to FINISH, a further subroutine call (this time to an output routine) is made. We then jump back to the start of the main block to collect another character.

Let's look at the difference between the JZ AND THE JP instructions and the 6502's BEQ. The first two result in jumps to addresses that have been specified by a two byte operand. The 6502, on the other hand, executes a 'branch if equal to zero' instruction, using a form of addressing known as relative addressing. The value of the operand is a one byte displacement, not an address. The branch is limited to values that can be specified in one byte. (Your assembler will calculate the displacement and should tell you if you exceed this limit.)

Relative addressing has the advantage of only requiring a two byte instruction (which makes for faster execution). Since we do not use an absolute address it also means that the code produced is relocatable. The disadvantage is that you are limited to a displacement of +127 to −128 (added to the contents of the program counter). We'll wait until we have examined two's complement arithmetic for a full explanation of this instruction.

To finish your program you will need to look at your machine manual again.

 Mar 84 5(3) 5 of 6.

Programs operating in a CP/M environment can use a JMP 0 instruction to 'reboot' the operating system. In such a case FINISH would look like this:

FINISH: JMP 0 ; Reboot operating system

Other systems may expect your program to perform an additional subroutine return instruction. The 8080/Z80 mnemonics for this instruction are both RET; the 6502 uses RTS:

FINISH: RET (or RTS for the 6502) ; Return to O/S.

Input — output subroutines

Once again you'll be dependent on your operating system: you should find addresses for functions such as direct input and console output. These might be given abbreviations such as GETCHAR, CONIN and OUCH etc. If you can call these functions directly, all you need to do is add the necessary equate definitions into your 'set up' block.

A possible problem

Discovering that your operating system routine returns a zero value to the accumulator if no keyboard character is available, that is, it doesn't 'wait' for input, is a problem. In this case you'll need to create a 'wait for input' loop. The idea is the same for all three processors, so we'll just show the 8080 version:

```
INPUT$ROUTINE:
CALL SYSTEM$INPUT$ROUTINE
CPI 0
JZ INPUT[ROUTINE
RET
; system direct input
; is accumulator 0?
; no input, so wait
```

In this case you would use an equate directive in the set up block to define SYSTEM$INPUT$ROUTINE's address, and then place the above routine into your program. Remember that when you call a subroutine, the address of the next instruction is pushed onto the stack. By making the last instruction of the routine a RET (or RTS is the case of the 6502) that address is popped off the stack and replaced into the program counter. The next instruction to be executed after the subroutine is then the one that followed the subroutine call instruction.

Having explained how our example program operates, it's over to you to put it into practice. Don't undervalue the time you'll spend checking your assembler documentation and computer handbook: it's up to you to become familiar with the information that is there waiting for you to use it. One word of warning: don't get so engrossed in your practical work that you forget to buy next month's *APC*, when we'll take the next step in your journey into assembly language programming.

FULL LISTING 6502 VERSION

```
                      SET-UP-BLOCK
CARRIAGE$RETURN    EQU   13
INPUT$ROUTINE      EQU   FDF0H
OUTPUT$ROUTINE     EQU   FD10H
                   ORG   800H
                      MAIN-BLOCK
START:             JSR   INPUT$ROUTINE
                   CMP   #CARRIAGE$RETURN
                   BEQ   FINISH
                   JSR   OUTPUT$ROUTINE
                   JMP   START
                      END-BLOCK
FINISH             RTS
```

For your convenience, a version of this month's program for each processor is included in this article.

END

Mar 84 5(3) 6 of 6.

Teach yourself Assembler

This month we continue our definitive introduction to assembly language programming with Paul Overaa's discussion of the 'alternation' structured building block.

Last month we defined sequence and repetition, and illustrated the ideas with a short program. This month we'll define alternation, the third and last of our structured building blocks. Simple alternation is exemplified by structured Basic's IF — THEN — ELSE type of coding. We can indicate the essential features using flowchart and Warnier diagram forms (see Figs 1 & 2).

We call this form simple alternation in order to distinguish it from those cases involving more than two alternatives. We are implying, in both representations, that any necessary preselection processing will have been performed.

A choice has to be made between two sets of actions based on a specified condition. Simple or 'binary' alternation, as we have shown, represents the existence of two mutually exclusive operation subsets. The idea can be generalised to condition tests with *n* mutually exclusive outcomes. This leads to the corresponding existence *n* mutually exclusive operation subsets within the logical program description.

This month we give you an illustration of how 'alternation constructs' can be created when writing programs in assembly language. To keep to familiar ground, we'll examine a slightly more complex problem related to the 'collection of characters' program shown last month.

Problem & solution

To write a routine to collect input from a keyboard and differentiate between control and printable characters. The routine should end when CARRIAGE RETURN key is pressed. Other control characters less than ASCII value 32 are to be ignored although a warning 'Bleep' is to be given. All other input characters should be echoed to the VDU screen (see Fig 3).

The problem is well defined. We need some sort of input routine; we need to compare each collected character to see if its a carriage return, that is, at ASCII 13 character. If it isn't, we need to know if it's another control character or a character to be printed.

Let's look at the equivalent Warnier diagram representation (see Fig 4). Note that we identify mutually exclusive subsets of actions by using a O sign. Remember that only *one* of these subsets will actually be performed.

What does the diagram tell us? We collect a character using an input routine. If the input character is a carriage return then we exit from the routine. If not, we make a further test to identify whether it's a control character or one to be printed. Having performed one of two possible alternative sets of actions, we return for another input character.

How would we program this in Basic? Here's a translation of the Warnier diagram in a Microsoft Basic form:

```
10   X$="A" must force an entry into
     WHILE/WEND LOOP
20   WHILE ASC (X$) — 13'<>0
```

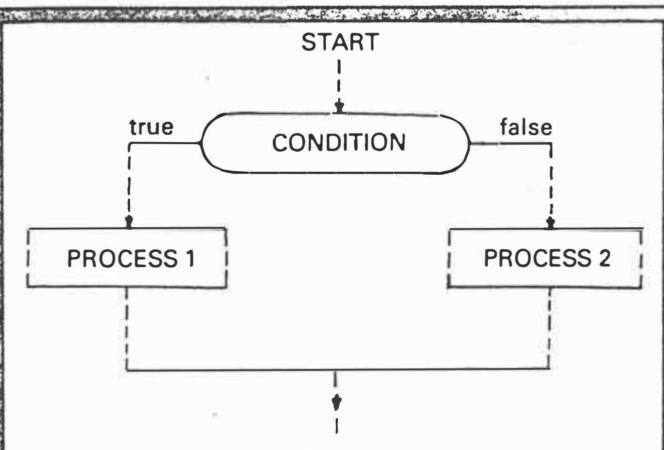


Fig 1. Simple alternation Flowchart Form

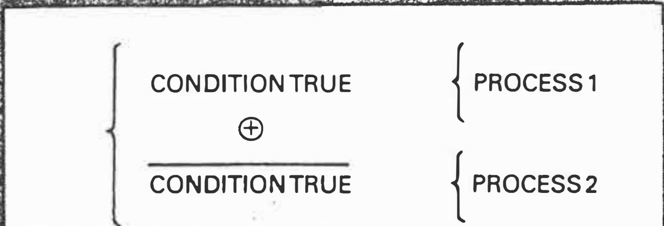


Fig 2. Simple alternation Warnier Form

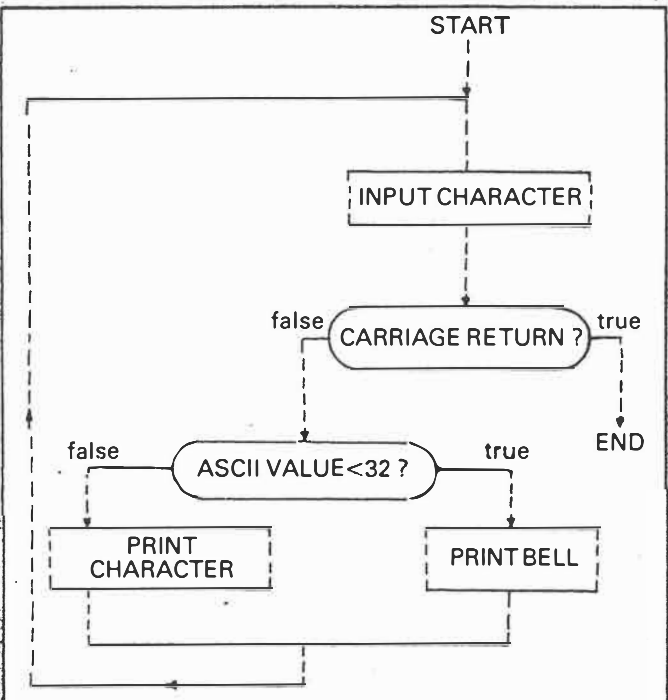


Fig 3. Flowchart representation for the example program

Apr 84 5(4) 1 of 5.

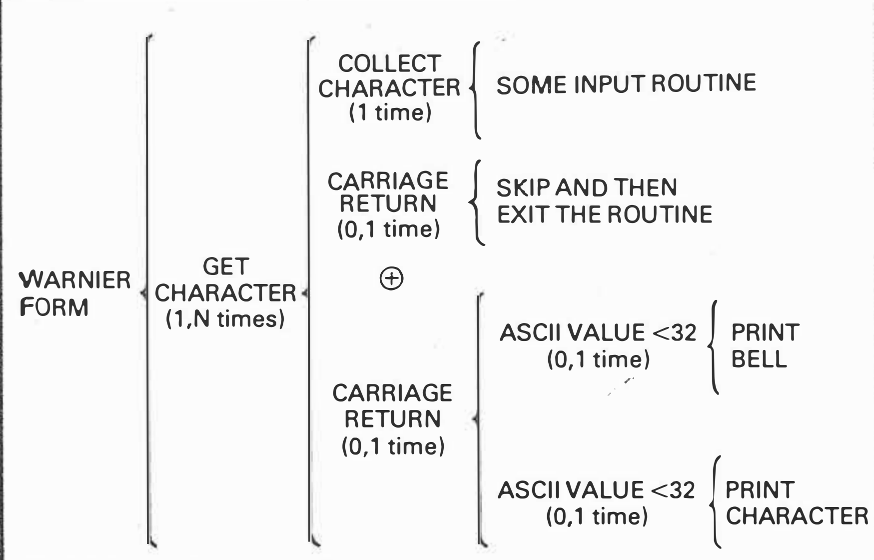


Fig 4. Warnier diagram for the example program

```
30  GOSUB 1000 ' some input routine
    collects input in X$
40  IF ASC (X$) <32 THEN GOSUB
    2000 ELSE GOSUB 3000
50  WEND
60  END
```

Subroutine 2000 will perform those actions concerned with 'printing a bell', and subroutine 3000 will concern itself with printing a character.

An equivalent form, and one that in terms of coding is arguably more efficient, can be obtained by using GOTO:

```
10  GOSUB 1000' some input routine
    collects input in X$
20  IF ASC (X$) = 13 THEN END
30  IF ASC (X$) <32 THEN GOSUB
    2000 ELSE GOSUB 3000
40  GOTO 10
```

Such a form is perfectly acceptable and shows a correct use of GOTO. It's only when they are used incorrectly that they create 'tangled' code that is difficult to maintain, difficult to understand and prone to errors.

You should not be misled into thinking that because a language is called unstructured, it is not possible to write well structured programs using that language.

Let's look at general ideas. We'll move on to the practical solution of our problem later. We're dealing with a particularly simple form of alternation, which is usually coded in its own special and very simple way.

Carry flag

Last month we used 'immediate' comparison instructions to test for equality. The instructions used were the 8080's CPI operand, the Z80's CO operand and in the case of the 6502 we used CMP # operand. When the contents of the accumulator are the same as the immediate byte specified, then the internal subtraction that occurs during the comparison results in the zero flag being set.

When these comparison instructions are used, several other flags are affected. Our present concern is the effect of these operations on the carry flag. We can tabulate all possible outcomes of such testing as in Fig 5.

In all cases, the contents of the accumulator, and of the immediate byte value specified, are treated as simple binary data.

We use the carry flag to detect control characters. For the purposes of our example, we define a control character as one having an ASCII code of less than 32.

8080 form

Let's look at the main part of an 8088 assembly language interpretation of these forms (Fig 6) and make some observations.

The 8080 mnemonics CC and CNC stand for 'call on carry' and 'call on not carry' respectively. They illustrate the concept of a conditional subroutine call, whose function is to perform the specified subroutine call, but only if the necessary flag condition is satisfied.

A more efficient form of coding is shown in Fig 7. It's more compact and satisfies the requirements of our problem, but you'll see later that problems can occur which, at present, are not immediately obvious.

We use an input routine to collect a character: this is required in the accumulator register. The CPI instruction compares the value of CARRIAGE$RETURN (which will have been previously set to 13 by an EQU directive), to the ASCII value of the character present in the accumulator. If the character present in the accumulator is a carriage return the zero flag will be set. As in the first form, the JZ instruction following this means we exit from the routine as soon as a carriage return character is detected.

If the character being looked at is not a carriage return, then we compare the

CONDITION	CARRY FLAG	ZERO FLAG
Accumulator > Immediate Byte	Cleared	Cleared
Accumulator = Immediate Byte	Cleared	Set
Accumulator < Immediate Byte	Set	Cleared

Fig 5. Carry flag operations

8080 — VERSION — 1

```
START:          CALL  INPUT$ROUTINE            ;Character in accumulator
                CPI   CARRIAGE$RETURN          ;End of input if true
                JZ    FINISH
                CALL  NOT$CARRIAGE$RETURN
                JMP   START                    ;Loop back for next character
NOT$CARRIAGE$
  RETURN:       CPI   SPACE
                CC    CONTROL$CHARACTER
                CNC   PRINTABLE$CHARACTER
                RET
```

Fig 6. 8080 interpretation

8080 — VERSION — 2

```
START:          CALL  INPUT$ROUTINE            ;Character in accumulator
                CPI   CARRIAGE$RETURN          ;End of input if true
                JZ    FINISH
                CPI   SPACE
                CC    CONTROL$CHARACTER
                CNC   PRINTABLE$CHARACTER
                JMP   START                    ;Loop back for next character
```

Fig 7. Efficient 8080 coding

accumulator contents to the value SPACE (again previously defined by using an EQU directive). *If the character present in the accumulator has an ASCII value less than 32, that is, if it's a control character, then the 8080's carry flag will be set. Otherwise the carry flag will be clear.*

In both of these examples, we are using the carry flag to implement the equivalent of an IF — THEN — ELSE structure. If the carry flag is set we do one set of operations; if the carry flag is not set we perform the alternative set of operations. The only necessary stipulation is that the status of the flag being tested must be preserved by the first of the subroutines to be called.

Control$character subroutine

This has to output a bell character. On most termianls this is done by sending the ASCII bell character to the terminal. In Basic you use PRINT CHR$ (7): in assembler a register is loaded with the value 7 and then the system output routine is used to send the character to the terminal. The normal procedure is to define BELL by an equate pseudo operation and the example shown assumes that this has been done:

8080 VERSION

```
CONTROL$
  CHARACTER: MVI   A,BELL
             CALL  OUTPUT$ROUTINE
             RET
```

To load the accumulator we are using the instruction 'MVI A,data': this is an example of a 'move immediate' instruction. The data byte following the op code is transferred to the specified register. We are specifying the accumulator, but it's also possible to use the instruction to load either B,C,D,E,H or the L registers. Note that we have now seen two immediate load 8080 instructions. MVI is used to load 8-bit values into a selected register. LXI is used to load a 16-bit value into a selected register pair (we used it last month to set up the stack pointer).

Printable$character subroutine

This simply has to output the character present in the accumulator, that is, it's your system output character routine.

Z80 form

The Z80 has conditional subroutine call capability similar to the 8080 processor. The syntax expected for the conditional subroutine calls is slightly different but this does not affect the essential ideas. We'll give the equivalents of the two versions of the 8080 code (Fig 8) and then explain why the version 1 forms have possible advantages.

Z80 VERSION 1

```
START:         CALL  INPUT$ROUTINE          ;Character in accumulator
               CP    CARRIAGE$RETURN        ;End of input if true
               JP    Z,FINISH
               CALL  NOT$CARRIAGE$RETURN
               JP    START                  ;Loop back for next character
NOT$CARRIAGE$
  RETURN:      CPI   SPACE
               CALL  C,CONTROL$CHARACTER
               CALL  NC,PRINTABLE$CHARACTER
               RET
```

Z80 VERSION 2

```
START:         CALL  INPUT$ROUTINE          ;Character in accumulator
               CP    CARRIAGE$RETURN        ;End of input if true
               JP    Z,FINISH
               CP    SPACE
               CALL  C,CONTROL$CHARACTER
               CALL  NC,PRINTABLE$CHARACTER
               JP    START                  ;Loop back for next character
```

Fig 8. Z80 forms for 8080 code

The Z80 has equivalent instructions to load a specified register with an 8-bit data value. The mnemonic LD, when used in the form 'LD register, 8-bit data value', is representing an instruction identical to the 8080's MVI. (When used in the form 'LD register pair, 16-bit data value' it is equivalent to the 8080's LXI instruction).

Z80 VERSION

```
CONTROL$
  CHARACTER: LD    A,BELL
             CALL  OUTPUT$ROUTINE
             RET
```

The LD mnemonic is, however, also used to represent register loading operations other than the loading of immediate data values. Some have 8080 equivalents that use different mnemonics, some do not have 8080 equivalents at all. Bear in mind for now the main distinction, *viz:* MVI and LXI on the 8080 are register loading instructions that use immediate addressing; that is, the operand is the bytes in memory that follow the op code (the instruction byte itself). LD on the Z80 processor, when used as shown, is using immediate addressing: additionally, it's used to represent data transfer using other addressing modes.

Go back now and look at the flowchart we are using for the example program.

6502 VERSION

```
START:          JSR   INPUT$ROUTINE
                CMP   #CARRIAGE$RETURN
                BEQ   FINISH
                JSR   NOT$CARRIAGE$RETURN
                JMP   START
NOT$CARRIAGE$
  RETURN:       CMP   #SPACE
                BCC   N$C$R$1
                JSR   CONTROL$CHARACTER
                RTS
N$C$R$1:        JSR   PRINTABLE$CHARACTER
                RTS
```

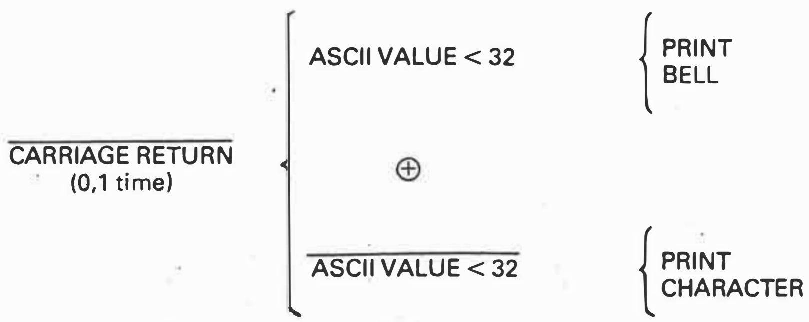


Fig 9. LD on the Z80 processor

 Apr 84 5(4) 3 of 5.

Can you pick out the subset of actions associated with 'not finding a carriage return character'? You'll probably agree that even in this simple example, the isolation of such subsets is not particularly obvious.

Try to find the same subset on the Warnier diagram. (Remember that we write the logical opposite of a statement by placing a bar over the statement.) The subset we are discussing is shown in Fig 9.

The reason we're particularly interested in this subset is because the 8080 and Z80 first versions explicitly treat the coding involved as a distinct subset; that is, actions corresponding to 'not carriage return' were implemented as a 'called subroutine'. The code is therefore related to the design diagram on this basis: the action subset is defined by coding as a subroutine. The advantage is that the structure of the diagram and the coding is isomorphic (a word used by mathematicians to imply structural similarity.)

The coding in the second version performs the same function as the coding in the first, but the action subset 'not carriage return' is not explicitly defined in the second form of code. The difference may not be immediately apparent to you, so let's briefly digress to explain this point.

There is a real advantage, especially when writing large assembly language programs, in being able to easily locate the section of the code that is relative to a particular action subset in the corresponding design diagrams. Such advantage is paid for by a slightly increased program size.

Hardcore assembly language programmers often take great exception to 'wastage of bytes' in this manner, and for certain applications their objections are justifiable. Our defence in general terms is two-fold. Firstly, it's often of great practical advantage to have isomorphic coding with the design diagrams. Secondly, memory is getting cheaper but debugging is not. Explicit subset definition based on isomorphism between the design diagram and the actual program code contributes in practice to significantly reduced debugging time. The message is simple — save bytes by all means, but distinguish carefully between pointless inefficiency and the deliberate choice of using a few more bytes to create code that can easily be compared to the design diagrams.

6502 form

Our 6502 processor cannot perform conditional subroutine calls. We must therefore find a way of creating such a facility. One fairly obvious solution involves using the 6502's relative branch instructions to select an appropriate subroutine (several of these are available). Since we're using the carry flag to detect control characters we can use the 'relative branch on carry clear' (whose mnemonic is BCC). With only slight rearrangement we can also use the complementary test BCS, which is the 'branch on carry set' instruction. The bulk of the code shown (see previous page) should be familiar, the differences are due only to the absence of conditional subroutine calls on the 6502.

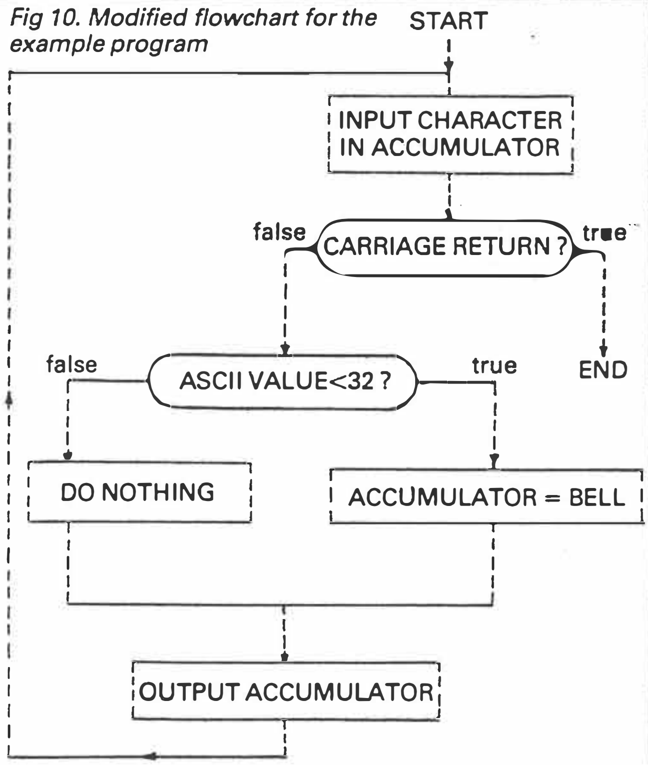
Don't be fooled by the presence of two RTS instructions — only one will actually be performed; that is, if the carry is set then the conditional relative branch is not performed, so we can execute the CONTROL$CHARACTER subroutine followed by a return instruction. If the carry is clear, the relative branch is performed, then PRINTABLE$CHARACTER is performed followed by the alternative return instruction.

6502 VERSION

```
CONTROL$
  CHARACTER:   LDA  # BELL
               JSR  OUTPUT$ROUTINE
               RTS
```

The CONTROL$CHARACTER subroutine in 6502 form is similar in principle to both the 8088 and the Z80 forms, we simply load the accumulator with the BELL character. The mnemonic used is LDA# data (the '#' sign is a 6502 mnemonic convention that indicates the operand is to be obtained from the next byte in memory; that is, it's signifying an 'immediate addressing' mode.

Fig 10. Modified flowchart for the example program



8080 PRACTICAL SOLUTION

```
START:              CALL INPUT$ROUTINE
                    CPI  CARRIAGE$RETURN
                    JZ   FINISH
                    CPI  SPACE
                    JNC  PRINT$CHARACTER
                    MVI  A,BELL
PRINT$
  CHARACTER:        CALLOUTPUT$ROUTINE
                    JMP  START
```

Z80 PRACTICAL SOLUTION

```
START:              CALL INPUT$ROUTINE
                    CP   CARRIAGE$RETURN
                    JP   Z,FINISH
                    CP   SPACE
                    JP   NC,PRINT$CHARACTER
                    LD   A,BELL
PRINT$
  CHARACTER:        CALLOUTPUT$CHARACTER
                    JMP  START
```

6502 PRACTICAL SOLUTION

```
START:              JSR  INPUT$ROUTINE
                    CMP  # CARRIAGE$RETURN
                    BEQ  FINISH
                    CMP  # SPACE
                    BCC  PRINT$CHARACTER
                    LDA  # BELL
PRINT$
  CHARACTER:        JSR  OUTPUT$ROUTINE
                    JMP  START
```

Teach yourself Assembler

Practical solutions

We have used our example to explain some general ideas. There is a very good reason why you would not, in practice, actually need to write subroutine based code for this particular example. Look back at some of the coding and think how we output printable characters, and how we output the ASCII bell character. In practice, we'll be using the accumulator to output the printable characters: we'll also use the accumulator to output the bell character. We will also, in both cases, be using our system OUTPUT$ROUTINE to send the character to the terminal.

The practical implementation of our problem has a certain amount of common ground that has not been used in our earlier general discussions. We now consider and modify our flowchart/design diagrams in the light of the above information (see Fig 10).

When we consider fully the practical implementation of our problem we see that one of the alternation subsets is a 'do nothing' process.

This type of structure is frequently handled by simple in-line conditional relative branching or conditional jumping. Based on a condition, we either perform some section of code, or avoid it by jumping over it. Bear in mind that this type of structure is a subclass of the simple alternation we first dealt with. There are still, from a theoretical viewpoint, two sets of actions. The distinction is that one of the subsets is an 'empty set'.

Having possibly struggled through some of the ideas we have presented so far, you will no doubt be pleased to see the assembly language code that results from our most recent efforts. If you've persevered up to this point, you should find the code fairly straightforward. In all three cases, the label PRINT$CHARACTER identifies the location to be jumped or branched to if the carry flag is not set.

You should now appreciate the relationship between simple alternation where two subsets of actions are involved, and the specific case of simple alternation where one of those subsets is an empty set. This month we've shown some of the ways in which the corresponding code can be writen.

Last word

The design of our solutions is derived from the logical examination of the problem.

The logical solution exists as an independent entity, and by having such solutions available before you start coding you will side-step many problems that other approaches walk straight into.

Using this approach, we find that we're left with the much smaller problem of how to use an available instruction set to implement an already known logical solution. We would like you to think about the implications (and in particular the benefits) of having language independent solutions available before coding is started.

If last month's 'main block' is modified to incorporate this month's practical solutions you should be able to run a version of the given problem; you might also like to experiment with some of the other ideas we considered.

Apr 84 5(4) 5 of 5.

Teach yourself Assembler

Addressing refers to how we specify the location of the operand, or, the byte or bytes upon which the instruction will operate. This month we look briefly at some of the addressing modes you need to be familiar with.

Each of the processors we are using has instructions to enable specified internal registers to be incremented or decremented. As an example, the 6502 uses INX to increase the value of the X register by one. The instruction when assembled results in a single object code byte. The 'address' of the operand (which in this case is the X register) is specified within the 'op code'. This form of addressing is termed 'implied' or 'implicit'. It is used in instructions such as register-to-register transfers and register increment/decrement.

If an instruction uses *immediate addressing*, it gets its operand byte/s from the location or locations immediately following the op code in memory. One example is in the loading of constant values into registers or register pairs.

These instructions, when assembled, result in two bytes of object code being produced — the op code followed by the data value. As we have seen previously, the 8080 and Z80 also have instructions that load register pairs with 16 bits of data, resulting in three bytes of object code being produced when the instructions are assembled — the op code byte plus the two data bytes.

Absolute addressing specifies a memory byte using a full 16-bit address. Such instructions, three bytes long, consist of the op code followed by the two byte address giving the location of the operand. POKE address, value . . . is a typical 'absolute addressing' Basic statement.

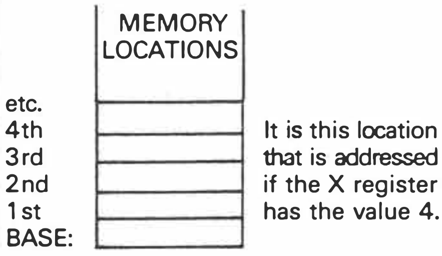
In the case of *relative addressing*, instead of an address we give a displacement to be added to the value already in the program counter. Such displacements are restricted on 8-bit micros because they have to be specified with one byte.

Up to now, the addressing modes we have looked at may be regarded as 'static', or to put it another way, once the program has been written the memory locations upon which the various instructions will operate are fixed, completely defined by the instructions you have selected. *Computed addressing* enables the address of an operand to be computed at run time and falls into two categories — indexed and indirect addressing. This month we look at indexed addressing and give you an idea of its usefulness.

Indexed addressing

Indexed addressing uses an address that is obtained by modifying a specified 'base address' given in the program. The 6502 load accumulator instruction LDA has several forms of addressing options including indexed. The mnemonic form LDA address X is an example of absolute indexing using the X register. The effect is to get the value present in the X register and add it to the specified base address. The base address is specified by you at assembly time in the same way that you specify an ordinary 'absolute' address, but the X register can be used by the program to compute the offset during program execution.

As an example, suppose that you have a table of 20 data items held in memory and have labelled the lowest byte location BASE (think of them as being 'numbered' from zero to 19). The instruction LDA BASE,X will access the base value if X is zero, the byte above this if X is one, and so on. In general, it will access the X'th data item of the table:



You've probably used similar ideas in your Basic programs, for example, FOR 1% = 1 TO 9: PRINT X (1%):NEXT 1%. When 1% = 4 you are referencing X(4). Indexed addressing is particularly useful for accessing successive data elements from tables or blocks of data. On the 6502 both the X and the Y registers are available as 8-bit index registers. The limitation on the 6502 is that X and Y are 8-bit registers, so the indexing offset is restricted. The 8080 processor has no indexing facilities at all. The Z80 has two 16-bit index registers but these are used to hold the base addresses, not the offset values.

Connect Four game

Let's illustrate indexing by examining one way to represent the game Connect Four. The essential details of the game are that two players have sets of coloured counters which are dropped (one at a time by alternate players) into one of seven columns. The first player to get four counters in a vertical, horizontal, or diagonal line wins the game. We want to look at how such a game can be represented within a computer and restrict ourselves to some simple beginnings:

1) Write a subroutine to set up (clear) the board representations.
2) Write a subroutine for players' moves (column number).
3) Write a subroutine to check that move is valid.
4) Write a subroutine to make the move on the computer's boards.
5) Write a subroutine to identify change of player for next move.

To define how we are to represent the game internally, each player will be represented on a separate board created by seven bytes of memory. Each byte will therefore constitute one column of the games board: bear in mind that the boards are 'twisted sideways in memory'. The base locations we have labelled are the 'column 0' bytes. As the game is played, column 0 is on the left hand side, column 6 on the right (Fig 1 should help you get the general idea). We've numbered the seven columns from 0 to 6 because of the way we'll use indexing to access them. The six rows, however, have been numbered from 1 to 6 because the row number then represents the 'bit position' within the byte.

The presence of a counter in a certain position will be indicated by setting the equivalent bit to 1. Our bytes are eight bits wide and we'll use the inner six bits of the bytes. We'll also select one byte of memory to act as a player switch, and change its value with each move to identify which player is making a move. Seven bytes will be used to count how

May 84 5(5) 1 of 5

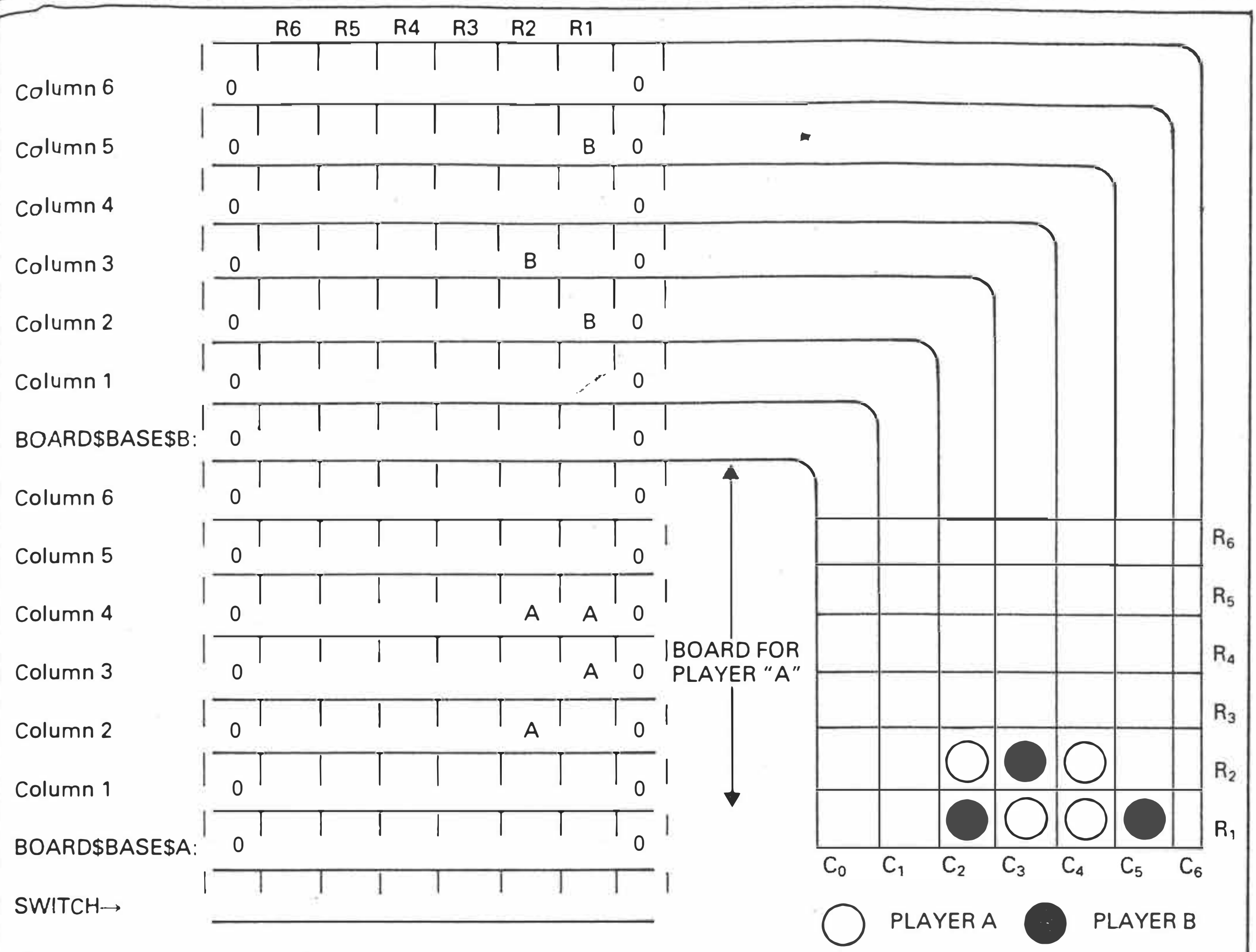


Fig 1 How layout of boards in memory relates to the normal 'playing' position of boards in practice.

many 'pieces' have been placed in a given column, and a further seven bytes used to identify the position of the last piece placed in a given column.

We'll discuss the overall ideas in terms of 6502 coding, but the layout of the boards and the general principles will be similar on the Z80; differences will be discussed, together with any changes needed after each individual subroutine discussion.

No indexing facilities are available on the 8080, so we must look at ways to create equivalent effects without indexed addressing.

Clear memory subroutine — 6502

We will, at the end of a finished program, use an assembler pseudo operation to reserve certain memory locations for use by our program: the operation is usually called 'reserve data storage space'. Our assemblers use the letters DS N to reserve N memory locations, and in our case, this space will 'sit' immediately above the actual program code.

We must write a subroutine to clear the area of memory assigned for the boards, and make the initialisations needed to switch byte (we'll arbitrarily set to zero to indicate player 'A' and to FF hex to indicate player 'B'). We initialise the seven bytes starting at the location labelled ROW$POINTER$BASE so that they contain the binary value 00000001, and will be using an operation called a left shift to push those single bits from right to left as the game progresses.

We initialise an area of memory by loading the accumulator with the number we wish to store, loading an index register with the number of bytes to initialise and then using a loop that implements indexed addressing to store the contents of the accumulator. We decrease the index register by one each time we pass through the loop, repeating until the index value becomes zero. Bear

```
        LDA  #value     ;Value we wish to store
        LDX  #n         ;n is the offset value
START:  STA  BASE,X     ;This is the indexed addressing bit
        DEX             ;Decrease the value in X by 1
        BNE  START      ;Back for next byte if X<>0
        STA  BASE       ;This does the base location
```

Fig 2 Typical 6502 form

```
        LDA  #value     ;Value we wish to store
        LDX  #N         ;Number of bytes
START:  STA  BASE-1,X   ;This is the indexed addressing bit
        DEX             ;Decrease the value of X by 1
        BNE  START      ;Back for next byte if X<>0
```

Fig 3 Alternative 6502 form

BYTE 1 1st OP CODE	BYTE 2 2nd,OP CODE	BYTE 3 DISPLACEMENT	BYTE 4 LITERAL
DD hex	36 hex	number	value
(<—–op code	values—–>)		

Fig 4 Layout of indexed Z80 instructions

```
        LD   IX,BASE       ;Set up index register IX
        LD   C,n           ;Number of bytes
START:  LD   (IX+0),value  ;Value stored at address in IX
        INC  IX            ;Increase register IX by 1
        DEC  C             ;Decrease counter C
        JR   NZ,START      ;Back for next byte if C<>0
```

Fig 5 Z80 version 1

```
         LD   IX,BASE-1     ;Byte below base address
         LD   IH,TARGET+2   ;HL points to displacement
         LD   (HL),N        ;N is the number of bytes
TARGET:  LD   (IX+0),value  ;Run time modified displacement
         DEC  (HL)          ;Decrease displacement
         JR   NZ,TARGET     ;Back for next byte if displ.<>0
```

Fig 6 6502 variable displacement implementation

```
        LXI  H,BASE     ;Initialise base value
        MVI  C,number   ;Number of bytes
START:  MVI  M,0        ;Store "immediate value" at location HL
        INX  H          ;Increase HL
        DCR  C          ;Decrease counter C
        JNZ  START      ;Back for next byte if C<>0
```

Fig 7 8080 version

```
ACCUMULATOR     00111001   <——— ASCII "9"
OTHER BYTE      00001111   <——— "MASK"
                ————————
RESULT          00001001   <——— REAL "9"
                ————————
```

Fig 8 Effect on the ASCII code for the number 9

in mind that because we don't branch back once the index register has become zero, we must initialise the base location separately.

The arrangement in Fig 2 is fairly straightforward, but you may consider it more convenient if we handle the base location within the loop itself. In actual fact we can, by using a typical 'trick' — we reference the byte *below* the base. In practice, we make use of another facility of modern day assemblers: we can perform simple arithmetic operations on labels, addresses, and so on. In Fig 3, we use the instruction STA Base-1,X so that the base address refers to the byte below that, labelled BASE. In this case, we must set the X register to the number of bytes we wish to reference. The equivalent form of the first 6502 example is shown in Fig 3.

In our finished routines we use two loops, one to initialise the memory between the byte labelled COUNTERSINBASE and the top of board 'B' with zeros, the other to initialise the seven row-pointer bytes.

Clear memory subroutine — Z80/8080

Indexing on the Z80 is implemented somewhat differently to the 6502. The index registers IX and IY are used to hold base addresses and not offset values. The indexed instructions on the Z80 offer the inclusion of a displacement value within the mnemonic form of the instruction. As an example, the instruction LD (IX+number), value loads the memory location whose address is 'IX+number' with the specified value. When assembled in memory, the layout of the instruction is as shown in Fig 4.

Note that we have an instruction here with a two byte op code, resulting in a total instruction length of four bytes. Let's use this instruction to create a simple loop to store a constant value in a set of adjacent locations (see Fig 5).

You'll notice that within this loop we are essentially using the index register as a 'pointer' to the location in which we wish to store the data item. We are not using 'indexing' in the true sense of our original definition, but are effectively using the IX register to specify an address which is then used to store the data.

If we wish to implement the variable displacement found on the 6502, we use the HL register pair to 'point' to the byte holding the displacement, and modify it during execution by using a DEC(HL) instruction as shown in Fig 6.

The first Z80 example offers some insight into an equivalent 8080 version. On the 8080, the HL register pair are frequently called the 'primary data pointer', with instructions existing to retrieve/store data in memory at the location specified by the current contents of HL. The standard notation for 8080 assemblers is to use the letter 'M' to signify a byte whose address is specified by the current contents of the HL pair. Thus, MVI M,6 will store the value 6 at a location specified by an address in HL. The example in Fig 7 is a direct translation of the Z80 version and also uses the HL register pair to point to successive locations in memory. The mnemonic INX represents an 8080 register pair increment instruction. DCR, however, is a single register decrement.

We have given versions of the 'clear memory' subroutine for all three processors: each uses two loops to perform the initialisations shown in Fig 1. At the end of the Z80/8080 routines we also set B and D registers to zero.

Get move subroutine 6502/Z80/8080

We use a system input routine to collect a column number in the accumulator. One immediate problem is that the ASCII character codes for the numbers 0 to 9 on the keyboard are not the numeric values of the numbers themselves. The values are as follows:

DECIMAL	BINARY	ASCII VALUE
0	00000000	00110000
1	00000001	00110001
2	00000010	00110010
3	00000011	00110011
4	00000100	00110100
5	00000101	00110101
6	00000110	00110110
7	00000111	00110111
8	00001000	00111000
9	00001001	00111001

To convert the ASCII form to a real binary equivalent of the input number, we need to set the upper four bits of the ASCII form to zero. This can be accom-

May 84 5(5) 3 of 5.

6502	Z80	8080
JSR INPUT$ROUTINE	CALL INPUT$ROUTINE	CALL INPUT$ROUTINE
AND #0FH	AND 0FH	ANI 0FH
TAX	LD C,A	MOV C,A

Fig 9 Processor codes for results

```
          BIT  SWITCH    ;N flag set if B's move
          BPL  G$M$1     ;Branch if A's move
          CLC
          ADC  #7        ;Board B needs additional offset
G$M$1:    TAY            ;Board offset in Y now
```

Fig 10 Final accumulator value in Y register

```
BYTE . . . ROW$POINTER$BASE,X ....... 00000100←image ofthe new move
                                                in the accumulator
BYTE . . . BOARD$BASE$A,Y ............. 00000010←current column state
RESULT NEEDED IN ACCUMULATOR ... 00000110← required new state
```

Fig 11 Creating a new move

plished by using an 'AND' operation. Essentially, two bytes, one of which is the accumulator, are compared bit by bit. If both bits are set to 1 then the corresponding accumulator bit is set to 1, otherwise the accumulator bit is set to 0. Fig 8 shows the effect on the ASCII code of the number 9.

The value we compare against is often called a 'mask'. On the 6502, several addressing modes are available with the AND operation. We'll use an immediate addressing mode to compare the accumulator with 0F hex (00001111 binary). The mnemonic will thus take the form AND #0FH, with the '#' sign signifying the immediate addressing form. Having obtained a proper numeric representation of the input character, we store it in the X register by using a transfer to X register (TAX !) instruction. We then have the column number for the user selected column in the X register.

On the Z80 and 8080 we use similar AND operations to mask the upper four bits of the accumulator, but we'll use the C registers to store our results. The code for all three processors is shown in Fig 9.

Computing offset into board area 6502

The offset into the boards is dependent on whether player A or player B is being dealt with. We use the value held in the switch byte in conjunction with a 6502 instruction called BIT. This is similar to the AND operation, but the result of the ANDing is not stored in the accumulator. It does, however, affect the following flags: bit 7 is placed into the 'N' flag, the 'V' flag is set equal to bit six of the byte being tested and the 'Z' flag is set or reset depending on the result of the ANDing. It's a strange instruction but it turns out to be very useful. We'll use it to test bit 7 of our switch byte, to place bit 7 into the N flag. We can then use a 'branch on plus' conditional branch instruction to either add seven to the value present in the accumulator (so that offset refers to board B), or to avoid doing so. *Note: it is the contents of the byte labelled SWITCH that is being tested (illustrating an absolute addressing instruction).*

Accept for now that it's necessary on the 6502 to use a 'clear carry flag' CLC instruction before adding a number to the accumulator. The reasons will be explained later in the series when we look at arithmetic operations in detail. CLC combined with an 'add with carry' ADC instruction will result in a 'normal' addition. CLC followed by ADC #7 will therefore add seven to the value of the accumulator. The final value in the accumulator is either the offset required (column number) for the A board or the equivalent offset for board B (relative to the base BOARD$BASE$A). We copy this value into the Y register by the method shown in Fig 10.

Computing the offset into the board area Z80/8080

As one of several alternatives, we load the accumulator with the contents of the switch byte and then add the contents to itself. This sets or clears the sign flag which is then used to add, or not add, the offset for board B. We have chosen to store the result in the E register.

Check move is valid subroutine — 6502/Z80/8080

On most microprocessors it's possible to shift bytes and registers to the left or right. The 6502 has instructions to perform various shifts and we'll make use of the instruction ASL, which is an arithmetic shift left. Our row pointer bytes are initialised to the value 00000001 binary by the 'clear memory' coding. If we consider the effect on the accumulator we can describe the shift effect diagramatically:

```
00000001 ←—— initial value of
             accumulator
00000010 ←—— accumulator
             after one ASL
             instruction
00000100 ←—— accumulator
             after two ASL
             instructions
```

The bit at the right hand side is always set to zero, the bit on the left hand side is shifted into the carry. If we use the instruction ASL A then we perform the above shift on the contents of the accumulator.

We want to load the accumulator with any one of seven bytes, depending on the value of the X register. We can do that easily on the 6502 using indexed addressing. We use the instruction LDA ROW$POINTER$BASE,X followed by ASL,A to shift the contents of the accumulator to the left (think about this carefully if you find it difficult to 'picture'). The single bit, after this instruction has been performed, will be in the bit position corresponding to the bit position on the board to be updated for this move. This representation has been arranged for reasons that will now become clear. If it has been shifted to the bit 7 position, the move is illegal because the column already has six pieces. How can we tell? The ASL instruction on the 6502 affects the carry, the zero and the N flags. The N flag is used to determine the status of bit, because on the 6502 all data movement and arithmetic instructions will set the N flag to the value of bit 7. The type of coding we use is shown in the following example:

```
LDA  ROW$            ;
     POINTER$
     BASE,X          ;Get column image
ASL  A               ;Shift to left
```

The Z80 also has shift instructions available, and the instruction SLA A will shift the contents of the accumulator to the left. With the 8080, shifting as we have described is not available. We could use one of the 'rotate' instructions but these do not affect the sign flag (the bit7 flag). To overcome this problem, we choose instead to add the contents of the accumulator to itself. This produces the equivalent effect of a left shift which does affect the sign flag.

May 84 5(5) 4 of 5.

Teach yourself Assembler

Making the move subroutine — 6502

After the 'check move' subroutine has been performed we'll have an image of the new move held in the accumulator. The first step is to store the contents of the accumulator back in the location used in the 'check move' subroutine. We can do this easily by using a 'store accumulator' STA ROW$POINTER$BASE,X instruction. Following this, it's necessary to add the new move into the appropriate board column. Let's take a typical example to illustrate the effect we wish to obtain to 'create the new move' (see Fig 11).

Another logical function exists called OR, that tests the accumulator with another specified byte. It will set any accumulator bit to 1 if either or both respective bits in the accumulator *or* the other byte specified is set to 1.

The 6502 has an instruction called ORA which 'ORs' the accumulator with another specified byte. We're going to use the instruction in an indexed addressing form in order to OR the image of the current state of the column in question with the new move present in the accumulator. The updated column will then be replaced into its correct memory position by using the equivalent 'store accumulator' (STA) instruction. Having done this, we increase the value of the corresponding numerical count of the number of pieces in the column. This is achieved with a single indexed addressing instruction INC COUNTERINBASE,X which increments the value currently in memory. The combined code to store the new row position byte, create the new move in memory and update the numeric count is achieved as follows:

```
STA   ROW$POINTER$BASE,X
ORA   BOARD$BASE$A,Y
STA   BOARD$BASE$A,Y
INC   COUNTERS$IN$BASE,X
```

Making the move — Z80/8080

In the clear memory routines we set B and D registers to zero. Since the column number and board offset for a move are held in the C and E registers, it should be apparent that the value of the BC pair is C and the value of the DE pair is E. This has been arranged in order to use an instruction that will add BC or DE to the HL register contents. If we load HL with BOARD$BASE$A, then use the Z80 instruction ADD HL,DE (DAD D for 8080), we set HL to the value HL+DE. In our case (DE=E), we are adding the offset E to the base address in HL which creates the equivalent of an indexed addressing instruction.

Changing the 'Player' subroutine — 6502/Z80/8080

We change players by changing the value of the byte we have labelled SWITCH. We set it to zero when we perform the clearing of memory. After each move we want to change the value, so that it alternates. We have seen examples of AND and OR as logical functions: another logical function is called 'exclusive OR'. This is similar to the OR described earlier, except that if both bits being tested are high, that is, are 1, then the accumulator bit will be set to 0 and not 1.

It's indirect addressing next month plus full listings of all the Connect Four subroutines discussed here, and the main block coding needed to run the programs.

APC May 84 5(5)
5 of 5.
p 98.

LANGUAGES

TEACH YOURSELF ASSEMBLER

Paul Overaa completes his explanation of addressing with a look at the use of one address to 'point' to another. The three subroutines for last month's Connect Four game are also provided.

We can illustrate the general idea of indirect addressing with the following Basic example. You have a data file of one thousand items whose record lengths are 128 bytes long, and you wish to sort these items in order of bytes 6 to 20 of each record in order to perform processing.

An easy approach is to load just the fifteen bytes of interest from each record into a vector (one-dimensional array), INDEX$() and, in addition, create a 'tag vector', I%() to hold each record's 'record number'. Before sorting, I%() will contain the numbers 1 to 1000 in order. A sort is then performed and the I%() vector is rearranged to 'mirror' any physical (or logical) changes made in the index vector. After sorting, INDEX$() will be in the required order but INDEX$(5), for example, may not now relate to the 5th record of the data file. By searching through INDEX$() we effectively move through the data file in the sorted order but this is of little use unless we can access the corresponding data record. To do this, we use the 'tag' vector I%() that holds the corresponding original record numbers: the record number of the first record in the sorted order, whose index value is INDEX$(1), is found from I%(1). Similarly, the Xth item in the sorted order is obtained from I%(X).

We use the tag vector I%() to 'point' to the records in the data file. By using the Basic statement GET #1,I%(5) to obtain the fifth record in the new sorted order, we specify its address indirectly: in effect, the 'address' of the record in question is held in the variable I%(5).

Addressing an operand indirectly in an assembly language instruction is a similar exercise. We do not specify the operand's address, but rather the locations from which the address may be obtained. In the case of the Z80 and the 8080 processors, a form of indirect addressing known as 'register indirect' is available. It is a register pair, rather than a pair of memory locations, that holds the address of the operand.

On the 6502, the concept of 'zero page addressing' is used. 'Page zero' refers to the first 256 bytes of memory (addresses 0000 hex to 00FF hex), considered as a set of storage locations. A zero page address has the advantage that it can be specified with one byte (the high byte of the address will always be zero, and can be easily created as an 'implied high byte' by the processor).

Then, we could in theory use a zero page equivalent of Z80/8080 register indirect addressing. An indirect address held in a register pair of a Z80 processor would emulate an indirect address held in two bytes of zero page RAM on the 6502.

Things are slightly more complex because the 6502 does not, in general, implement simple indirect addressing. Instead, two forms of mixed 'indexed and indirect' addressing are available. One is called 'indirect indexed' and the other 'indexed indirect'. The single exception is the instruction JMP (address), which is a jump to the location specified by the contents of two bytes, address and address+1.

Indirect indexed

The 6502 uses the contents of the zero page byte specified within the instruction as the low order part of the indirect address. It also collects the contents of the next byte in the zero page and uses that as the high order part of the address. The indirect address obtained is then used as a base address for Y register indexing: that is, the contents of the Y register are added to the indirect address and it's this final addres that is used.

It may appear complicated as a single operation but it helps to consider the two stages as separate actions. The 'indirect bit' is simply the specifying and using of the zero page locations as a 'store' for the base address. Once this base address is available, the indexing is performed in just the same way as absolute indexing (described last month). The advantages are that we don't have to specify the base address at the time we write the program, and that we can, during execution of the program, modify the contents of the zero page bytes to 'point' to any number of different base addresses as required.

If we wish to load the accumulator with the contents of an indirect indexed specified byte, the instruction will take the form LDA (zero page address), Y. The zero page address specified is then used to obtain the base address for the indexing (the general idea can be seen in Fig 1). If the zero page bytes held the address corresponding to the byte labelled BASE, we would then access the Yth byte of the set BASE, BASE+1, BASE+2, etc.

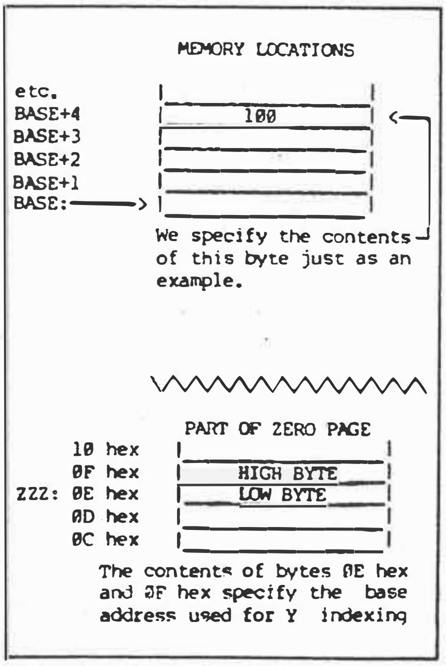


Fig 1 Obtaining the base address for indexing

If the Y register contained the value 4 then the instruction LDA (ZZZ),Y would result in the value 100 being placed in the accumulator.

Jun 84 5(6) 1 of 5

Indexed Indirect

This addressing mode uses the 6502's X register and performs the indexing first. In this case, a table or 'set' of addresses is held in the zero page. The X register provides the index offset from the base address and the contents of this byte, plus the contents of the succeeding byte which are used as an indirect pointer to another memory location. The type of instruction format required can be shown as follows: to load the accumulator, use LDA (zero page address,X); to 'OR' the accumulator, ORA (zero page address,X) should be used.

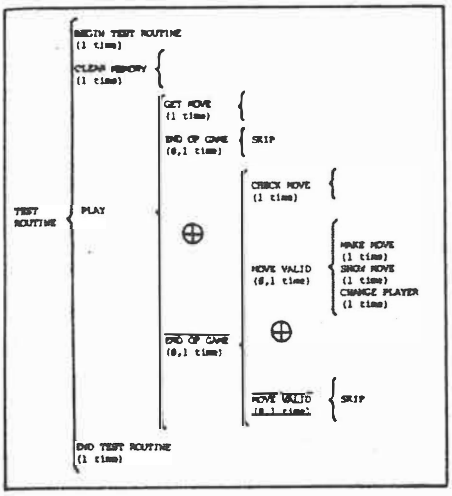


Fig 2 Test bed control routine

The requirement of a zero page address in both indexed indirect and indirect indexed addressing is a 6502 processor restriction and has nothing to do with the actual concepts of indirect addressing. Even bearing in mind such restrictions, you should be aware that the 6502 implementation of indirect addressing is substantially more powerful than the simple register indirect form available on the Z80 and 8080 processors.

Connect Four

Last month we developed routines applicable to the game 'Connect Four' (see Subroutines A, B and C). These are first steps in such a development, but even at this stage the routines must be checked to ensure they work. A common technique (and one that is frequently used) is to write short 'test bed' controller routines — short patches of code that use the subroutines under development in order to check their performance. To illustrate how we go about this we've written a routine to test the subroutines featured here. The first job is to sketch out a brief 'controller structure' using a Warnier diagram as shown in Fig 2.

Most of the statements in Fig 2 correspond to existing subroutines. The 'end of game' statements imply that we can detect the end of the game. This we cannot do since no playing strategy is available yet. With this in mind, we must be satisfied with either testing the routines by using an 'infinite loop', or terminating the controller program when a particular keyboard character is detected.

We choose the latter option and use a carriage return to signify the end of game condition. We also need a temporary 'show move' code, and for illustration purposes adopt a simple solution — output the row number representing the position in the given column that the latest move will occupy. In writing the controller routine the aim is only to test the subroutines we have written. The controller block starts by clearing the memory, then we collect a character with the 'get move' subroutine. If a carriage return is detected we end the program, otherwise we check the move. If the move is illegal (a move to a full column) we ignore it, otherwise we make the move on the internal boards and display it by outputting the 'row number'. Finally, we change the player before returning to collect another move.

We have not included a check to ensure that any column number entered lies between 0 and 6 as this method of identifying a move is only applicable during the development stage, where such checks are not absolutely necessary.

In all three cases we have kept the test bed program listings separate from the listings of the developed subroutines, making it easier to see the basic ideas behind the controller routine and also allowing us to view the subroutines 'in isolation'. If problems occur, one useful tip is to modify the controller routine to eliminate calls to any suspect sub-

```
              SET UP BLOCK Z80 VERSION

CARRIAGE$RETURN     EQU  13
OPERATING$SYSTEM    EQU  5              ;Entry point
                    ORG  100H
                    JP   STACK
                    ORG  150H
STACK:              LD   SP,$—2

           CONTROLLER ROUTINE Z80 VERSION

PLAY:        CALL CLEAR$MEMORY
             CALL GET$MOVE
             LD   A,C
             CP   CARRIAGE$RETURN
             JP   Z,FINISH              ;End of game
             CALL CHECK$MOVE
             JP   M,PLAY                ;Illegal move so
                                          ignore it
             CALL MAKE$MOVE
             LD   A,(HL)                ;Get row number
                                          for display
             OR   00110000B             ;Convert to ASCII
                                          equivalent
             CALL OUTPUT$ROUTINE        ;'Show move'

             CALL CHANGE$PLAYER
             JP   PLAY                  ;Back for next
                                          move
FINISH:      JP   0                     ;Re-boot
                                          operating
                                          system

          IN THIS AREA PLACE SUBROUTINES
                 TO BE TESTED
        (INCLUDE ANY I/O ROUTINES REQUIRED)

              WORKSPACE DEFINITIONS

ROW$POINTER$BASE:     DS   7     ;Bit marked
                                   'counter height'
COUNTERS$IN$BASE:     DS   7     ;Numeric form
                                   'counter height'
SWITCH:               DS   1     ;Identifies
                                   current player
BOARD$BASE$A:         DS   7     ;Player A's board
                                   bit map
BOARD$BASE$B:         DS   7     ;Player B's board
                                   bit map
```

Fig 3 Test bed program Z80 version

Jun 84 5(6) 2 of 5.

routines. To be safe, you may prefer to start with a controller routine that just calls the 'clear memory' subroutine. Once this is working satisfactorily the 'get move' subroutine can be included. In this way, the controller routine can be built up one piece at a time.

Internal boards

The internal representations of the boards may be examined in several ways. We might write a routine to display the contents of the bytes in binary form, use the system monitor to examine the bytes in question, or use a dynamic debugging tool (CP/M's DDT program, for example) that allows examination of memory areas during execution of a program. The binary display routine makes a useful exercise, and you may like to think about how it can be programmed. If you're not sure, have a look at the article on the Warnier techniques published in January issue. A memory dump routine was developed which gives plenty of clues.

The layout of the test bed program is equivalent in all three processors (see Figs 3, 4 and 5). We start with a 'set up' block — defining equates, initialising stacks, and so on as required. The controller routine comes next, which makes calls to the various subroutines that have been developed. Immediately following this we place the subroutines we wish to test, including any other necessary routines: for example, any input/output routines needed. Lastly, we identify our data storage areas which 'sit' on top of the program.

An error crept into Fig 5 of last month's article.
The 6502 carry flag is CLEARED when the A register is < compared value. The BCC operands in the 6502 routines should therefore be changed to BCS.

```
SET UP BLOCK 8080 VERSION

CARRIAGE$RETURN     EQU  13
OPERATINGSSYSTEM    EQU  5           ;Entry point
                    ORG  100H
                    JMP  STACK
                    ORG  150H
STACK:              LXI  SP,$—2

CONTROLLER ROUTINE 8080 VERSION

        CALL CLEAR$MEMORY
PLAY:   CALL GET$MOVE
        MOV  A,C
        CPI  CARRIAGE$RETURN
        JZ   FINISH              ;End of game
        CALL CHECK$MOVE
        JM   PLAY                ;Illegal move so
                                   ignore it
        CALL MAKE$MOVE
        MOV  A,M                 ;Get row number
                                   for display
        ORI  00110000B           ;Convert to ASCII
                                   equivalent
        CALL OUTPUT$ROUTINE      ;'Show move'
        CALL CHANGE$PLAYER
        JMP  PLAY                ;Back for next
                                   move
FINISH: JMP  0                   ;Re-boot
                                   operating
                                   system

IN THIS AREA PLACE SUBROUTINES
TO BE TESTED
(INCLUDE ANY I/O ROUTINES REQUIRED)

WORKSPACE DEFINITIONS

ROW$POINTER$BASE:    DS  7    ;Bit marked
                                'counter height'
COUNTERS$IN$BASE:    DS  7    ;Numeric form
                                'counter height'
SWITCH:              DS  1    ;Identifies
                                current player
BOARD$BASE$A:        DS  7    ;Player A's board
                                bit map
BOARD$BASE$B:        DS  7    ;Player B's board
                                bit map
```

Fig 4 Test bed program 8080 version

```
SET UP BLOCK 6502 VERSION

CARRIAGE$RETURN     EQU  13
INPUT$ROUTINE       EQU  0FD1BH
OUTPUT$ROUTINE      EQU  0FDEDH
                    ORG  6000H
```

```
CONTROLLER ROUTINE 6502 VERSION

        JSR  CLEAR$MEMORY
PLAY:   JSR  GET$MOVE
        TXA
        CMP  #CARRIAGE$RETURN
        BEQ  FINISH                  ;End of game
        JSR  CHECK$MOVE
        BMI  PLAY                    ;Illegal move so
                                       ignore it
        JSR  MAKE$MOVE
        LDA  COUNTERS$IN$BASE,X      ;Get row number
                                       for display
        ORA  #00110000B              ;Convert to ASCII
                                       equivalent
        JSR  OUTPUT$ROUTINE          ;'Show move'
        JSR  CHANGE$PLAYER
        JMP  PLAY                    ;Back for next
                                       move
FINISH: JMP  0                       ;Re-boot
                                       operating
                                       system

IN THIS AREA PLACE SUBROUTINES
TO BE TESTED
(INCLUDE ANY I/O ROUTINES REQUIRED)

WORKSPACE DEFINITIONS

ROW$POINTER$BASE:    DS  7    ;Bit marked
                                'counter height'
COUNTERS$IN$BASE:    DS  7    ;Numeric form
                                'counter height'
SWITCH:              DS  1    ;Identifies
                                current player
BOARD$BASE$A:        DS  7    ;Player A's board
                                bitmap
BOARD$BASE$B:        DS  7    ;Player B's board
                                bit map
```

Fig 5 Test bed program 6502 version

```
CLEAR$MEMORY: LD   IX,COUNTERS$IN$BASE
              LD   C,22
C$M$1:        LD   (IX+0),0          ;Set these bytes
                                       to 0
              INC  IX
              DEC  C
              JR   NZ,C$M$1
              LD   IX,ROW$POINTER$BASE
              LD   C,7
C$M$2:        LD   (IX+0),1          ;Set these bytes
                                       to 1
              INC  IX
              DEC  C
```


Jun 84 5(6) 3 of 5

```
         JR     NZ,C$M$2
         LD     B,0             ;We set B and D
                                  to 0 in order
         LD     D,0             ;to use ADD HL,
                                  BC, later
         RET
```

GET MOVE Z80 VERSION

```
GET$MOVE:    CALL  INPUT$ROUTINE
             AND   0FH           ;Mask upper four
                                   bits
             LD    C,A           ;Save column
                                   no. in C register
             LD    E,A           ;and as the board
                                   'A' offset
             LD    A,(SWITCH)
             ADD   A
             JP    M,G$M$1
             LD    A,E           ;Get column
                                   number back
             ADD   7             ;Board 'B'
                                   additional offset
             LD    E,A           ;Replace offset
                                   value in E
G$M$1:       RET
```

CHECK MOVE Z80 VERSION

```
CHECK$MOVE:  LD    HL,ROW$POINTER$BASE
             ADD   HL,BC         ;Effective HL+C
                                   since B=0
             LD    A,(HL)        ;Image of
                                   column's last
                                   move
             SLA   A             ;Left shift
             RET
```

MAKE MOVE Z80 VERSION

```
MAKE$MOVE:   LD    (HL),A        ;Replace
                                   updated column
                                   image
             LD    HL,BOARD$BASE$A
             ADD   HL,DE         ;Now HL points
                                   into boards
             OR    (HL)          ;Create new
                                   board image
             LD    (HL),A        ;and replace in
                                   memory
             LD    HL,COUNTERS$IN$BASE
             ADD   HL,BC         ;HL now points to
                                   count byte
             INC   (HL)          ;Increase
                                   numeric count
             RET
```

CHANGE PLAYER Z80 VERSION

```
CHANGE$PLAYER:  LD    A,(SWITCH    Get current
                                     player
                CPL                ,Complement
                                     the 'switch' byte
                LD    (SWITCH),A   ;Changed for
                                     next player
                RET
```

Subroutine A Clear memory Z80 version

```
CLEAR$MEMORY: LXI   H,COUNTERS$IN$BASE
              MVI   C,22
C$M$1:        MVI   M,0          ;Set these bytes
                                   to 0
              INX   H
              DCR   C
              JNZ   C$M$1
              LXI   H,ROW$POINTER$BASE
              MVI   C,7
C$M$2:        MVI   M,1          ;Set these bytes
                                   to 1
```

```
         INX    H
         DCR    C
         JNZ    C$M$2
         MVI    B,0             ;We set B and D
                                  to 0 in order
         MVI    D,0             ;to use DAD
                                  instructions later
         RET
```

GET MOVE 8080 VERSION

```
GET$MOVE:    CALL  INPUT$ROUTINE
             ANI   0FH           ;Mask upper four
                                   bits
             MOV   C,A           ;Save column
                                   no. in C register
             MOV   E,A           ;and as the board
                                   'A' offset
             LDA   SWITCH
             ADD   A
             JM    G$M$1
             MOV   A,E           ;Get column
                                   number back
             ADI   7             ;Board 'B'
                                   additional offset
             MOV   E,A           ;Replace offset
                                   value in E
G$M$1:       RET
```

CHECK MOVE 8080 VERSION

```
CHECK$MOVE:  LXI   H,ROW$POINTER$BASE
             DAD   B             ;Effective HL+C
                                   since B=0
             MOV   A,M           ;Image of
                                   column's last
                                 move
             ADD   A             ;Effective left
                                   shift
             RET
```

MAKE MOVE 8080 VERSION

```
MAKE$MOVE:   MOV   M,A           ;Replace
                                   updated column
                                   image
             LXI   H,BOARD$BASE$A
             DAD   D             ;Now HL points
                                   into boards
             ORA   M             ;Create new
                                   board image
             MOV   M,A           ;and replace in
                                   memory
             LXI   H,COUNTERS$IN$BASE
             DAD   B             ;HL now points to
                                   count byte
             INR   M             ;Increase
                                   numeric count
             RET
```

CHANGE PLAYER 8080 VERSION

```
CHANGE$PLAYER:  LDA   SWITCH       ;Get current
                                     player
                CMA                ;Complement the
                                     'switch' byte
                STA   SWITCH       ;Changed for
                                     next player
                RET
```

Subroutine B Clear memory 8080 version

```
CLEAR$MEMORY: LDX   #22
              LDA   #0           ;Set these bytes
                                   to 0
C$M$1:        STA   COUNTERS$IN$
                      BASE—1,X
```


Jun 84 5(6) 4 of 5

```
            DEX
            BNE   C$M$1
            LDX   #7
            LDA   #1                  ;Set these bytes
                                        to 1
C$M$2:      STA   ROW$POINTER$
                    BASE—1,X

            DEX
            BNE   C$M$2
            RTS
```

GET MOVE 6502 VERSION

```
GET$MOVE:   JSR   INPUT$ROUTINE
            AND   #0FH
            TAX                       ;Column number
                                        (0-6) in X now

            BIT   SWITCH              ;N flag set if B's
                                        move
            BPL   G$M$1               ;(Branch if A's
                                        move!)
            CLC
            ADC   #7                  ;Board B needs
                                        additional
                                        offset
G$M$1:      TAY                       ;Board offset in Y
                                        now
            RTS
```

CHECK MOVE 6502 VERSION

```
CHECK$MOVE: LDA   ROW$POINTER$        ;Image of
                    BASE,X            column's last
                                        move

            ASL   A                   ;Shift contents to
                                        left
            RTS                       ;'N' Flag set if
                                        illegal
```

MAKE MOVE 6502 VERSION

```
MAKE$MOVE:  STA   ROW$POINTER$        ;Replace
                    BASE,X              updated

            ORA   BOARD$BASE$A,Y      ;Create new
                                        board image
            STA   BOARD$BASE$A,Y      ;and replace in
                                        memory
            INC   COUNTERS$IN$        ;Increment
                    BASE,X              numeric count
            RTS
```

CHANGE PLAYERS 6502 VERSION

```
            LDA   SWITCH              ;Get current
                                        player
            EOR   #0FFH               ;Complement
                                        the 'switch' byte
            STA   SWITCH              ;Changed for
                                        next player
            RTS
```

Subroutine C Clear memory 6502 version

Jun 84 5(6) 5 of 5.

TEACH YOURSELF ASSEMBLER

Paul Overaa continues his series on assembly language programming with a general discussion of arithmetic operations.

This is part five of APC's Teach Yourself Assembler series. It's unique in using Basic as its point of reference, and avoiding the 'drop you in it' approach often used on this subject. Three processors, the Z80, 6502 and 8080 are covered in detail, but enough information is provided to enable users of other processors to follow the course. Copies of earlier articles in the series, which started in March 1984, may be obtained from our Back Issues dept.

The 8-bit processors, such as the Z80, 8080 and 6502, have instructions to perform only elementary addition and subtraction. To provide anything more sophisticated requires us to program the more complex procedures in terms of these simple operations. This month we look at some general ideas, then next month we'll relate this to assembly language routines.

We 'take for granted' the facilities offered by high level languages for adding, subtracting, multiplying and dividing, and an appreciation of how languages, such as Basic, actually perform the 'arithmetic' is useful for gaining insight into the problems involved when providing such facilities. Our first job is to look, in a general sense, at the way we represent numbers inside a computer.

Integers

With the eight bits of a single byte we can represent numbers from 00000000 Binary to 11111111 Binary — that is, from 0 to 255 decimal. To represent larger numbers we must use 'more bits'. By using two bytes for the representation we can deal with integer numbers up to the value 65536 (1111 1111 1111 1111 Binary). The magnitude of a number that can be represented in this way is therefore limited by the number of bytes we choose to assign to its representation. This form of representation is called 'unsigned binary'. To allow for the occurrence of negative numbers it is necessary to make provisions within the representation of the number to indicate whether it is positive or negative. This can be done by using one bit as a 'sign' bit. By convention, we use the most significant bit, the left-hand bit. It is set to zero to represent a positive number and to 1 to indicate a negative number. An 8-bit 'signed binary' number will therefore have only seven bits for the numerical value. For example, Decimal 5, which is 101 Binary, can be represented as follows:

+5 Signed binary form = 0 0000101
−5 Signed binary form = 1 0000101
^
|

(Leading bit used to represent the sign of the number — separated for clarity only.)

By using a suitable number of bytes, and using one bit as a sign bit, we can represent both positive and negative numbers of any magnitude. Are our problems of representation over? If we just wanted to represent the numbers, then yes. The problem is that we want to manipulate them (add, subtract, and so on). We'll first add two positive numbers, 4 and 5, as an example:

+4 is 00000100
+5 is 00000101
Result 00001001 represents 9 (which is correct).

Now we try adding the two numbers −4 and +5:

−4 is 10000100
+5 is 00000101
Result 10001001 represents −9 (which is incorrect).

The correct result is +1, so clearly a problem exists with the representation, or the way we are using it. The solution lies in using 'two's complement' representation. In this form, positive numbers are represented in the usual signed binary form. The difference lies in the representation of the negative numbers. We take the 'unsigned binary' form and complement it: turn all the 1s into 0s and 0s into 1s (often called the 'ones' complement' form). Having done this, we add 1 to the result to obtain the final 'two's complement' representation. It can be shown that by using this representation, the results of arithmetic operations, including the sign, come out correctly.

Here are some examples to outline the general idea. Let's try the addition of −4 to +5 again. +5, being a positive number, is represented in usual signed binary form but we must convert −4 to its two's complement in the manner described above. We represent the number in binary form, complement it, and add 1 to the complement. When the correct representation has been obtained, retry the example and check the result. The details are shown in Fig 1.

One of the 'rules' of two's complement arithmetic is that the setting of the carry flag can safely be ignored.

If the magnitude of a result is too large to be expressed within the bits allotted for the representation of the numerical part of the number, it's possible for the sign bit to be changed accidentally. This is called 'overflow' and the effect is an incorrect result.

The most obvious cause of such an error is an 'internal carry' from bit6 to bit7, as the following example will show:

0 0111111 two's comp form of +63
0 1000001 two's comp form of +65
1 0000000
↑

(The 'sign' bit has been changed due to a carry from bit 6 to bit 7).

Overflow can also occur when we add two negative numbers. In general, it occurs when the result cannot be expressed in the seven bits available. It is obviously useful to be able to detect such a condition and most processors, including the Z80 and 6502, have an 'overflow' flag

Jul 84 5(7) 1 of 3

for this purpose (the 8080 does not possess an overflow flag).

Multiple-byte integers

The magnitude of the largest integer we

Conversion to the two's complement form

00000100 is binary 4

11111011 One's complement form of –4

11111100 Two's complement form of –4

Addition of the two's complement forms

11111100 –4 (two's complement form)

00000101 +5 (two's complement form)

(1) 00000001 result +1 (which is correct)

|

Carry flag is set in this example

Fig 1 Addition using two's complement arithmetic

```
 4 REM
 5 REM
 6 REM
10 INPUT Please enter integer value; X%          Input an integer value
20 MSB$=HEX$(PEEK(VARPTR(X%)+1))'               Most significant byte
30 IF LEN(MSB$)=1 THEN MSB$=0+MBS$
40 LSB$=HEX$(PEEK(VARPTR(X%)))'                 Least significant byte
50 IF LEN(LSB$)=1 THEN LSB$=0+LSB$
60 PRINT MSB$+LSB$'                             Shows how X% is stored
70 END
80 REM
```

Fig 2 Print hex representation of integer X%

	< MSB	>	< LSB	>
66 =	0000	0000	0100	0010 Binary
Complement	1111	1111	1011	1101
Add 1				1
Two's complement form	1111	1111	1011	1110
Equivalent Hex form	F	F	B	E

Fig 3 Explanatory details for –66

1 bit Sign	n bits Exponent	1 bit Sign	m bits Mantissa

Fig 4 Schematic form

APC Jul 84 5(7) p. 63.
2 of 3.

can represent is governed by the number of bytes used. We can show this by looking at how Microsoft's Basic stores the 'integer variables'. When you write the Basic statement LET X% = 10, the per cent sign indicates that an integer variable, x%, is being assigned the value 10. Can we write a program to look at the internal representation of such a number? Yes, easily.

The function VARPTR(X%) is used to obtain the address of the variable X%. This byte, and the contents of the following byte, are examined using the PEEK() function (after prior translation to hexadecimal form by use of the HEX$() function). For hex numbers less than 16, the HEX$() function returns only one character (for example, F rather than 0F), so we add the '0' to such numbers from within the program. The program in Fig 2 asks for an integer value and prints the hex form of the internal representation.

(Note: The function VARPTR(), an abbreviation of 'variable pointer', is normally used to pass addresses of variables from a Basic program to an assembly language routine).

If this program is run with the number 15, 000F will be obtained, which corresponds to the binary number 0000 0000 0000 1111. With −66, you will get FFBE — Fig 3 shows the reason why.

Floating point representation

The representation of wide ranges of decimal numbers has its own special problems. The usual way of coping with wide variations in magnitude is to use scientific notation. For example, 26063.15 can be represented as 2.606315×10^4, or −0.000003415 can be written -3.415×10^{-6}. This gives a clue to providing a similar computer representation. We need to reserve bits for the mantissa, and further bits for the exponent. We also need to indicate the signs of each part of the number. In scientific notation, we 'normalise' the number by moving the decimal point to a position where the mantissa takes a value between 1 and 9.999. It transpires that for floating point representation, it's better to move the 'binary point' to the far left of the number:

111.1101 is represented as $.1111101 \times 2^3$

.0000111 is represented as $.111 \times 2^{-4}$

The general floating point format is based on a schematic form, m and n varying according to the number of bits chosen. Fig 4 illustrates the essential idea.

Binary coded decimal

For some applications, it is necessary to have complete numerical accuracy. An often quoted example is the use of computers in accountancy. For these applications, an alternative representation called 'binary coded decimal', or 'BCD', is sometimes used.

The principle is to code each digit separately, using as many bits as necessary. Each digit requires four bits with some combinations being unused:

BCD	*Number*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010 – 1111	Unused codes

Two digits are packed into each byte, thus the amount of space a number will require is dependent on how many characters are present.

The advantage of representing numbers in this way is that complete accuracy is obtained. The disadvantages are firstly, that more memory is required to store the numbers and secondly, that arithmetic operations are slower.

Next month: Having briefly described some of the more common ways of representing numbers within a computer, we turn our attention to simple routines that use some of the forms we have discussed. In the meantime, try this experiment: take a number and multiply is by 2, 4 and 8. Express the number and all the products in their binary form. What do you notice about the bit patterns?

END

APC Jul 84 5(7) p 64.
3 of 3.

TEACH YOURSELF ASSEMBLER

Paul Overaa discusses the arithmetic operations of addition, subtraction, multiplication and division on the 6502, Z80 and 8080 processors.

This is part six of APC's Teach Yourself Assembler series. It's unique in using Basic as its point of reference, and avoiding the 'drop you in it' approach often used on this subject. Three processors, the Z80, 6502 and 8080 are covered in detail, but enough information is provided to enable users of other processors to follow the course. Copies of earlier articles in the series, which started in March 1984, may be obtained from our Back Issues dept.

The basic arithmetic instructions available on the 8080, Z80 and 6502 processors are for addition and subtraction. The 6502 operates on 8-bit operands only, but both the 8080 and Z80 have certain instructions that enable 16-bit operands to be dealt with.

Addition Z80

On the Z80, addition instructions take the form ADD A, operand. The specified operand is added to the value present in the accumulator, and in symbolic form we can write A ← A + operand. Various forms of addressing are possible, as follows:

ADD A,8: adds the immediate value 8 to the accumulator — that is, is performing the function A ← A + 8.

ADD A,B: adds the contents of the B register to the accumulator, thus performing the function A ← A + B.

ADD A,(HL): adds to the accumulator the contents of the byte whose address is specified by HL — that is, A ← A + (HL).

ADD A,(IX+d): in the indexed addressing form, the address of the byte to be added is found by adding a specified displacement to the address held in index register IX. The symbolic representation is A ← A + (IX+d).

Instructions for 16-bit operations use HL, IX or IY as destination registers. Typical examples are as follows:

ADD HL,DE; adds the contents of the DE pair to the contents of HL, thus performing HL ← HL+DE.

ADD IX,BC: in a similar fashion, this adds the contents of BC to the index register IX.

On the Z80, the instruction 'add with carry' (ADC) will include, in the 'addition', the carry flag value: ADC A,B will perform the function A ← A + B + Carry. The usefulness of this instruction can be seen from the example in Fig 1. We add two 'two byte numbers' — 255 and 257 — by adding the two low bytes first and then adding the two high bytes.

The addition of the low bytes causes a 'carry' to occur: the ADC instruction takes it into account when the high bytes are added. As a general rule, multibyte addition is performed by using a normal addition instruction for the first (least significant) bytes, and using the 'add with carry' instructions for succeeding bytes. The program in Fig 2 adds the contents of two 'two byte numbers' held in locations labelled FIRST$NUMBER and SECOND$NUMBER.

Because of the existence of double register addition instructions, it's possible to write a much simpler 16-bit addition program on the Z80. DE and HL can be loaded directly with the numbers to add, and an ADD HL,DE instruction used to perform the 16-bit addition with one addition instruction (Fig 3).

Addition 8080

Immediate loading of 8080 register pairs uses a LXI instruction. LXI H, SECOND$NUMBER will load the HL pair with the 16-bit address equivalent to the label SECOND$NUMBER. LDA is a direct loading of the accumulator from the byte whose address is FIRST$NUMBER. 'M' is the 8080 assembler convention to specify an

```
HIGH BYTES              LOW BYTES
0000  0000              1111  1111          00FF hex = Decimal 255
0000  0001              0000  0001          0101 hex = Decimal 257
----------              ----------
0000  0010     (1)      0000  0000     ←    Result of addition
   ↑            ↑           ↑               Low byte addition causes
   |            |                           the carry to be set
   |        Carry flag
   |
High byte addition
with ADC includes
the carry value
```

Fig 1 Z80 'add with carry' instruction

```
LD    HL,SECOND$NUMBER      ;HL points to low byte of second number
LD    A,FIRST$NUMBER        ;Get low byte of first number in Acc
ADD   A,(HL)                ;Add low bytes
LD    (RESULT),A            ;Store low byte of result
LD    A,FIRST$NUMBER+1      ;Get high byte of first number
INC   HL                    ;Now points to high byte of second number
ADC   A,(HL)                ;Add high bytes + carry
LD    (RESULT+1),A          ;Store high byte of result
```

Fig 2 Z80 16-bit addition

```
LD    DE,(FIRST$NUMBER)     ;Load DE with first number
LD    HL,(SECOND$NUMBER)    ;Load HL with second number
ADD   HL,DE                 ;Performs HL ← HL + DE
LD    (RESULT),HL           ;Store result
```

Fig 3 Z80 alternative 16-bit addition

```
LXI     H,SECOND$NUMBER      ;HL points to low byte of second number
LDA     FIRST$NUMBER         ;Get low byte of first number in Acc
ADD     M                    ;Add low bytes
STA     RESULT               ;Store low byte of result
LDA     FIRST$NUMBER+1       ;Get high byte of first number
INX     H                    ;Now points to high byte of second number
ADC     M                    ;Add high bytes + carry
STA     RESULT+1             ;Store high byte of result
```

Fig 4 8080 16-bit addition

```
LHLD  FIRST$NUMBER     ;Load HL with first number
XCHG                   ;Swap to DE
LHLD  SECOND$NUMBER    ;Load HL with second number
DAD   D                ;Performs HL ← HL + DE
SHLD  RESULT           ;Store result
```

Fig 5 8080 alternative 16-bit addition

```
CLC                          ;Clear carry flag
LDA     FIRST$NUMBER         ;Low byte of first number
ADC     SECOND$NUMBER        ;Add low bytes
STA     RESULT               ;Store low byte of result
LDA     FIRST$NUMBER+1       ;High byte of first number
ADC     SECOND$NUMBER+1      ;Add high bytes
STA     RESULT+1             ;Store high byte of result
```

Fig 6 6502 16-bit addition

```
LD      HL,SECOND$NUMBER     ;HL points to low byte of second number
LD      A,FIRST$NUMBER       ;Get low byte of first number in Acc
SUB     (HL)                 ;Subtract low bytes
LD      (RESULT),A           ;Store low byte of result
LD      A,FIRST$NUMBER+1     ;Get high byte of first number
INC     HL                   ;Now points to high byte of second number
SBC     A,(HL)               ;Subtract high bytes with borrow
LD      (RESULT+1),A         ;Store high byte of result
```

Fig 7 Z80 16-bit subtraction

```
LD      DE,(FIRST$NUMBER)    ;Load DE with first number
LD      HL,(SECOND$NUMBER)   ;Load HL with second number
AND     A                    ;Clear the carry flag
SBC     HL,DE                ;Equivalent to HL ← HL + DE
LD      (RESULT),HL          ;Store result
```

Fig 8 Z80 alternative 16-bit subtraction

```
LXI     H,SECOND$NUMBER      ;HL Points to low byte of second number
LDA     FIRST$NUMBER         ;Get low byte of first number in Acc
SUB     M                    ;Subtract low bytes
STA     RESULT               ;Store low byte of result
LDA     FIRST$NUMBER+1       ;Get high byte of first number
INX     H                    ;Now points to high byte of second number
SBB     M                    ;Subtract high bytes with borrow
STA     RESULT+1             ;Store high byte of result
```

Fig 9 8080 16-bit subtraction

```
SEC                          ;Set carry flag
LDA     FIRST$NUMBER         ;Low byte of first number in accumulator
SBC     SECOND$NUMBER        ;Subtract low bytes
STA     RESULT               ;Store low byte of result
LDA     FIRST$NUMBER+1       ;High byte of first number in accumulator
SBC     SECOND$NUMBER+1      ;Subtract high bytes
STA     RESULT+1             ;Store high byte of result
```

Fig 10 6502 16-bit subtraction

indirectly addresed memory location, and it refers to the byte whose address is contained in the HL register pair. Thus, ADD M on the 8080 is performing the same function as ADD A,(HL) on the Z80. STA is the 8080 'store accumulator direct', the contents of the accumulator are stored at the address specified. INX is a 'double register increment'. After the INX H instruction, HL is pointing to the byte after that labelled SECOND$NUMBER — that is, it is pointing to SECOND$NUMBER+1. Typical 8080 code is shown in Fig 4.

An equivalent version of the second Z80 form using the HL and DE register pairs can be written, the only difference being that on the 8080 it's not possible to load the DE pair directly. Instead, we load HL with the contents of the byte labelled FIRST$NUMBER, then use an exchange instruction XCHG to 'swap' the contents of the HL and DE registers. The first number is therefore placed into DE, leaving us free to re-load HL with the second number. A double register DAD D instruction is then used to perform the function HL ← HL+DE. The instruction SHLD will store the contents of the HL register pair in the two bytes RESULT and RESULT+1 (Fig 5).

Addition 6502

The only addition instruction available on the 6502 is an 'add with carry' (the mnemonic is ADC). This is no real disadvantage, but it does mean that if you wish to perform 'normal addition' you must 'clear' the carry flag before using ADC. The 6502 can be conditioned to operate in one of two modes, Binary or Decimal. The operations we are discussing are related to normal binary operation and we'll assume that the processor has been placed in binary mode by using a CLD (clear decimal mode) instruction (Fig 6).

Z80 subtraction

As with the addition instructions, it's useful to have two types of subtraction — normal subtraction and 'subtraction with borrow'. Normal subtraction (mnemonic SUB) is used for the 'low bytes' (least significant bytes), and subtraction with borrow (mnemonic SBC) is used for the succeeding bytes (most of the instructions in Fig 7 are identical to the earlier addition program). If, after the subtraction of the least significant bytes the carry flag has been set, this indicates that the value subtracted from the accumulator is greater than the accumulator value itself — a borrow has occurred. The SBC instruction allows for this 'borrow' by including the carry flag in the subtraction.

A more compact version using HL and DE can also be written. The only subtraction instruction available for the

2 of 5. Aug. 84 5(8)

double register operations is a subtract with carry. This being so, we clear the carry flag by ANDing the accumulator with itself, thus producing a 'normal subtraction' (there is no explicit 'clear carry Z80 instruction' that could be used). The code in Fig 8 gives the general idea.

Subtraction 8080

The mnemonics are SUB and SBB. The 8080 does not have double register subtraction instructions, and the example in Fig 9 uses the accumulator as in the first 8080 addition example.

Subtraction 6502

The 'subtract with borrow' instruction on the 6502 performs the function A ← A − operand − $\overline{\text{Carry}}$, with the bar over the carry indicating the 'complement' of the carry. Borrow is thus defined as the carry flag complemented. The 6502 equivalent for a 16-bit subtraction starts by SETTING the carry flag using a SEC instruction. As with Z80 and 8080 forms, the least significant bytes are dealt with first. The equivalent 6502 program for a 16-bit subtraction is shown in Fig 10.

These ideas can be expanded to any number of bytes and the general principles remain unchanged, but for now we'll turn our attention to the slightly more complicated problem of multiplication and division.

Multiplication

Consider the base 10 product shown below:

```
   2 5    ← Multiplicand
   1 2    ← Multiplier
 ------
  2 5     ← Partial products
   5 0
 ------
 3 0 0    ← Result
 ------
```

Let's take this simple product and do the same calculation using base 2 — that is, binary arithmetic:

```
          1 1 0 0 1   ← Multiplicand
                        (25)
            1 1 0 0   ← Multiplier
                        (12)
  -------------------
  1 1 0 0 1           ← Partial
                        products
    1 1 0 0 1
      0 0 0 0 0
        0 0 0 0 0
  -------------------
  1 0 0 1 0 1 1 0 0   ← Result (300)
  -------------------
```

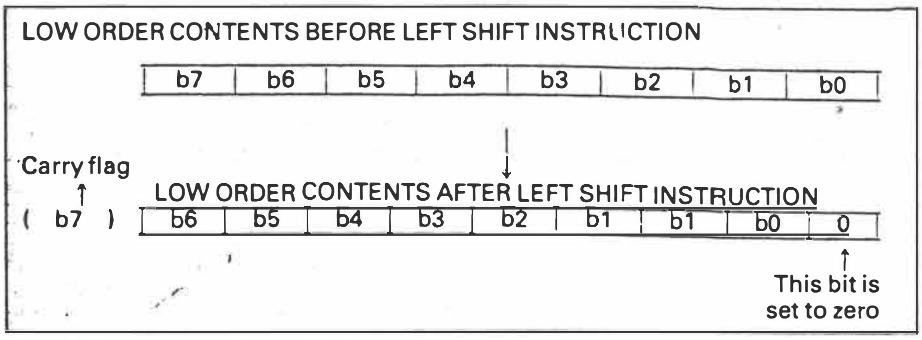


Fig 11 Normal left shift on low order byte

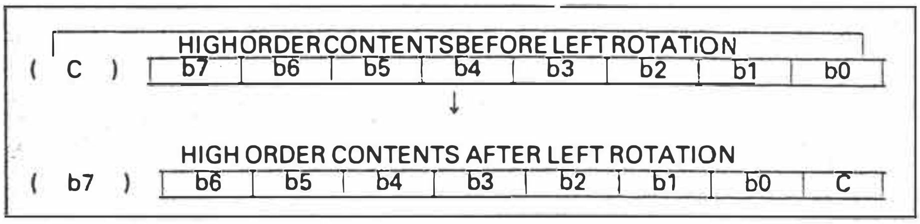


Fig 12 Rotation to the left

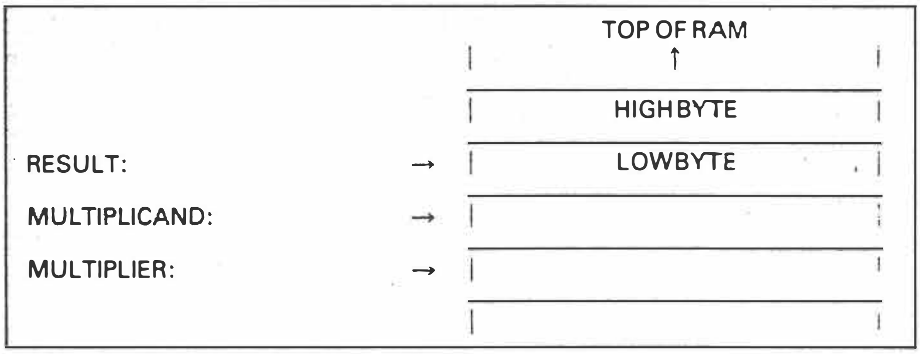


Fig 13 Layout in memory of 8-bit multiplication

The important point is that the partial products are either zeros, or a 'shifted' version of the multiplicand; we can use this knowledge to devise an algorithm for binary multiplication. For each 'Bit' in the multiplier, we ask: 'Is this bit set to 1?' If it is, we add the shifted equivalent of the multiplicand to the result. Two approaches are possible: we can either 'left shift' the multiplicand during the operations, or we can 'right shift' the bytes or registers that are storing the result.

Before showing some typical code for an 8-bit multiplication, we need to understand the general ideas behind creating '16-bit shifts'. Generally, the left shift operations available on our microprocessors will push bit7 into the carry flag. When attempting to left shift a 16-bit (2-byte) value, we can use a normal left shift on the low order byte as shown in Fig 11.

Bit7 falls into the carry flag, and to obtain a 16-bit shift we must shift this bit, now in the carry flag, into bit8 of the 16-bit number. In other words, we want to push this carry value into bit 0 of the high order byte. We need an instruction that performs a left shift and includes the carry, and the most commonly implemented instructions that perform this are rotation instructions. Rotation to the left has the effect shown in Fig 12.

By utilising a combination of left shift on the low order byte and a left rotation (through the carry) on the high order byte, we can left shift a 16-bit number held in two bytes or in two 8-bit registers; the principles can be extended to any number of bytes as required. Instructions are usually available for the equivalent right shifts and right rotations. Occasionally, you will find 'tricks' being used to create 16-bit left shifts. One favourite on the Z80 is to use the double-register addition instructions to add a register pair to itself. For example, ADD HL,HL results in a 16-bit arithmetic left shift.

Let's see how these ideas help to produce a simple multiplication program that takes an 8-bit number held in a location labelled MULTIPLICAND, mul-

3 of 5. Aug. 84 5(8)

```
          LD     HL,MULTIPLIER         ;HL points to multiplier
          LD     C,(HL)                ;Get multiplier in C register
          LD     B,8                   ;B is used as a 'bit' counter
          INC    HL                    ;Now HL points to
                                        multiplicand
          LD     E,(HL)                ;Get multiplicand in E
                                        register
          LD     D,0                   ;Now DE = multiplicand!
          LD     HL,0                  ;HL will be used to hold
                                        result
-----------------------------------------------------------------
MULTIPLY: SRL    C                     ;Least sig but (multiplier)
                                        into carry
          JR     NC,SKIP               ;Indicates least sig bit is
                                        zero
          ADD    HL,DE                 ;Add partial product to
                                        result
SKIP      SLA    E                     ;Left shift multiplicand low
                                        byte
          RL     D                     ;Left rotate high byte
                                        through carry
          DEC    B                     ;Decrease bit counter
          JP     NZ,MULTIPLY           ;Do next bit
          LD     (RESULT),HL           ;Store result
```

Fig 14 Z80 8-bit multiplication

```
          LD     HL,(MULTIPLIER—1)     ;Get multiplier in H register
          LD     L,0                   ;Clear to zero
          LD     B,8                   ;B is used as a 'bit' counter
          LD     DE,MULTIPLICAND       ;Get multiplicand in E
                                        register
          LD     D,0                   ;Now DE = multiplicand!
-----------------------------------------------------------------
MULTIPLY: ADD    HL,HL                 ;16-bit left shift
          JR     NC,SKIP               ;Indicates least sig bit is
                                        zero
          ADD    HL,DE                 ;Add partial product to
                                        result
SKIP:     DJNZ   MULTIPLY              ;Do next bit
          LD     (RESULT),HL           ;Store result
```

Fig 15 Z80 8-bit multiplication version two

```
          LXI    H,(MULTIPLIER—1)      ;Get multiplier in H register
          MVI    L,0                   ;Clear to zero
          MVI    B,8                   ;B is used as a 'bit' counter
          LXI    D,MULTIPLICAND        ;Get multiplicand in E
                                        register
          MVI    D,0                   ;Now DE = multiplicand!
-----------------------------------------------------------------
MULTIPLY: DAD    H                     ;16-bit left shift
          JNC    SKIP                  ;Indicates least sig bit is
                                        zero
          DAD    D                     ;Add partial product to
                                        result
SKIP:     DCR    B                     ;Decrease counter
          JNZ    MULTIPLY              ;Do next bit
          SHLD   RESULT                ;Store result
```

Fig 16 8080 8-bit multiplication

tiplies it by a second number held in location MULTIPLIER, and places the result into the two bytes starting from the lowest byte, which has been labelled RESULT (Fig 13).

Z80 multiply

The code in Fig 14 is split into two parts. Firstly, we load the registers with the following data: HL is loaded with the address of the multiplier, and register C is then loaded with the multiplier itself (using indirect addressing through HL). A 'bit count' of eight is loaded into the B register, and this will be used to count how many times we have gone through the 'multiplication loop'. The HL pair are then incremented so that they point to the multiplicand, which is placed in the E register using a LD E,(HL) instruction. Register D is set to zero because, although the multiplicand is only eight bits, we'll need 16 bits available as in the 16-bit left shift operation explained earlier. Finally, HL is set to zero and will be used to collect the result prior to storing it in locations RESULT and RESULT+1.

The second section of code is the actual multiplication. We use a right shift operation on the C register so that the least significant bit goes into the carry. This means that if the carry becomes 'set', then the least significant bit was a '1'. The carry flag is tested and if it has not been set, the partial product is zero and we skip the addition. Before moving on to the start of the loop again, the DE pair are shifted using a left shift followed by a left rotation, and the 'bit counter' B is decreased. If B is not zero we repeat the loop again, otherwise the final result is stored in RESULT and RESULT+1.

This 'first attempt' code can be shortened and improved in several ways. The Z80 has a combined 'decrement and relative jump on not zero' instruction. It operates using the B register as the counter and decreases the B register by 1, and if B<>0, the relative jump is performed. Another improvement is also possible, but is less obvious. If the Multiplier is placed

```
               LDA  #0
               STA  RESULT
LDX            #8
MULTIPLY:      LSR  MULTIPLIER
               BCC  SKIP
               CLC
               ADC  MULTIPLICAND
SKIP:          ROR  A
               ROR  RESULT
               DEX
               BNE  MULTIPLY
               STA  RESULT+1
```

Fig 17 6502 8-bit multiplication

4 of 5. Aug 84 5(8)

in the H register and the L register set to zero, the instruction ADD HL,HL will perform a 16-bit left shift. As the multiplier is shifted out during processing, we create room to store the result in HL.

To take advantage of this arrangement we must shift the multiplier to the LEFT, meaning that we deal with the most significant partial product first. We can also 'tighten up' the initial loading code by loading HL as a register pair starting one byte *below* the multiplier (so that the multiplier goes into the H register). The L register can be cleared after this 16-bit load in readiness for receiving the result. A similar 'trick' can be used to load the multiplicand into the E register.

These improvements have been made in the version shown in Fig 15.

Multiplication 8080

Translation to 8080 form is straightforward. All the improvements made in the second Z80 version can be implemented on the 8080 except for the automated DJNZ instruction. Relative jumps are not supported, so normal jump instructions are used in the loop (Fig 16).

6502 multiply

On the 6502, we cannot use any 16-bit 'paired registers', but we can create similar effects by considering the accumulator as the high byte of such a pair, and a memory location as the equivalent low byte. Such a combination can be shifted in the same way as explained earlier. The X register can be utilised as a 'bit counter', and an LSR (logical shift right) instruction can be used to push the least significant bits of the multiplier into the carry flag; this is used to decide whether or not to add the multiplicand.

In the example shown in Fig 17 the multiplicand is not shifted, it is just added to the accumulator. We right shift the 'accumulator memory byte' 16-bit pair using ROR instructions, and this provides an equivalent alternative.

Did you try the left shift experiment suggested last month? If you did, you will have found that shifting a number to the left is equivalent to multiplying the number by 2. Similarly, two left shifts are equivalent to multiplying by 4. In general, an 'n bit' left shift will multiply the value by 2 raised to the power 'n'. END

5 of 5 Aug 84 5(8)

TEACH YOURSELF ASSEMBLER

This month Paul Overaa continues his discussion of assembly language programming with a breakdown of the myriad and often complicated first steps.

> This is part seven of *APC*'s Teach Yourself Assembler series. It's unique in using Basic as its point of reference, and avoiding the 'drop you in it' approach often used on this subject. Three processors, the Z80 6502 and 8080 are covered in detail, but enough information is provided to enable users of other processors to follow the course. Copies of earlier articles in the series, which started in March 1984, may be obtained from our Back issues dept.

One of the problems with writing assembly language programs is that it's often difficult to know just where to begin. This is not so much an indictment of low level languages, but an indictment of many of the techniques used to identify the first steps needed. To give an example of how such breakdown can be performed, let's look at the simple problem of storing text in a buffer area.

Buffers

It is often necessary to temporarily store an input item before using it. Such temporary storage areas are termed buffers, and are areas of memory that we reserve as part of our program/memory use strategy. We select an arbitrary but commonly used arrangement that will take one page (256 bytes) of memory. The first byte, byte 0, will hold the character count; the remaining bytes will hold the characters typed in at the keyboad. A schematic description is shown in Fig 1.

In the source code, such an area would be defined using one of the 'define space' directives. The conventions vary from assembler to assembler but our Z80 assemblers, for example, would use the pseudo-op — DS 256 to reserve 256 bytes of uninitialised space within the object code.

What do we need to implement a routine that will place a word in a buffer? Obviously, some type of 'loop' (*cf* repetition structure) and a means of counting the number of characters typed in are required. We also need to test for the end of a word. Normally, we use a carriage return (ASCII 13) to signify the end of input, and earlier on in the series we used several loops that tested for such a character. We must also be able to identify which location in our buffer area is to be used for the current input character.

In the June issue we talked of 'computed addressing', that is, indexed and indirect addressing. We use computed addressing to determine the address of a 'buffer' pointer', to tell us where in the buffer the next character should be placed. On the 8080 we can only use indirect addressing, and we simply load the HL register pair with the address of the start of the buffer and increment HL as we add characters. On the Z80 and 6502 we can use either indirect or indexed addressing, which brings us to the following question. Can you see why it's better to use indirect addressing on the Z80, yet on the 6502 indexed addressing is more suitable?

The Z80 indexing facilities use a fixed displacement. Unless we create a run time modified displacement (which is of no real benefit in this case), it's simpler to use the HL register pair as an 'indirect pointer' into the buffer area (we'll need to maintain a 'count' of the number of characters). The 6502, on the other hand, implements a form of indexing whose displacement is held in the X or Y registers. By using this arrangement, we won't need to maintain a separate character count as the indexing variable itself provides the count.

We can define the essential characteristics of a 'Get-word' subroutine with the diagram in Fig 2. With one important (and deliberate) omission, this diagram will provide the overall structure needed.

What does the diagram show? After some initialisation (for example, setting up pointers) we perform a routine 'Build-string' at least once and up to a maximum of n times. The purpose of Build-string is to use a system routine to collect a character; then, if the character is *not* a carriage return we increment the character count and place the new character in the buffer. As soon as we detect a carriage return, we exit from Build-string and perform the last operation of the most left-hand side bracket: that is, END GET WORD. This entails

COUNT	AREA FOR TEXT STORAGE
Byte 0	← Bytes 1 to 255 →

Fig 1 Text buffer layout

Sep 84 5(9) 1 of 4

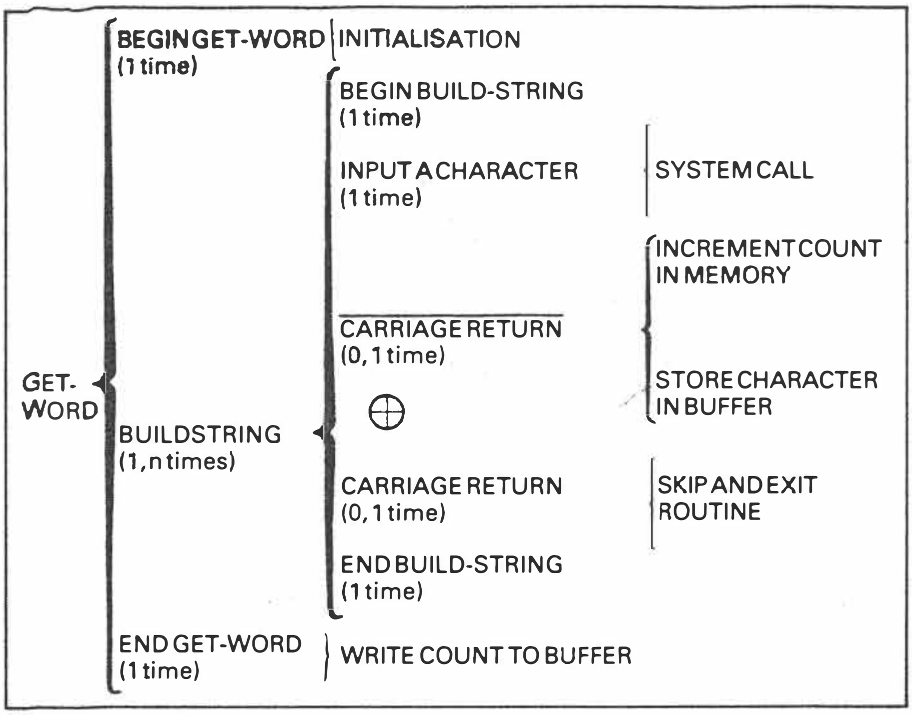


Fig 2 Input requirements for Get-word subroutine

```
GET$WORD:       LD    C,0                 ;Initialise count
                LD    HL,BUFFER$SPACE     ;Start of buffer

BUILD$STRING:   CALL  INPUT$ROUTINE       ;System call
                CP    CARRIAGE$RETURN     ;Is it a CR?
                JR    Z,CLOSE$BUFFER
                INC   C                   ;Increment count
                INC   HL                  ;Increment pointer
                LD    (HL),A              ;Store character
                JR    BUILD$STRING        ;Back for next character

CLOSE$BUFFER:   LD    HL,BUFFER$SPACE     ;Need start address again
                LD    (HL),C              ;Store character count
                RET                       ;Return from subroutine
```

Fig 3 Get-word Z80 version one

```
GET$WORD:       MVI   C,0                 ;Initialise count
                LXI   H,BUFFER$SPACE      ;Start of buffer


BUILD$STRING:   CALL  INPUT$ROUTINE       ;System call
                CPI   CARRIAGE$RETURN     ;Is it a CR?
                JZ    CLOSE$BUFFER
                INR   C                   ;Increment count
                INX   HL                  ;Increment pointer
                MOV   M,A                 ;Store character
                JMP   BUILD$STRING        ;Back for next character


CLOSE$BUFFER:   LXI   H,BUFFER$SPACE      ;Need start address again
                MOV   M,C                 ;Store character count
                RET                       ;Return from subroutine
```

Fig 4 Get-word 8080 version one

writing the character count at the head of the buffer. A Z80 translation is shown in Fig 3 using a simple loop. When a carriage return is detected, we perform a relative jump to CLOSE$BUFFER, re-load HL with the starting address of the buffer, and store the contents of the C register (which is used to hold the character count) by using a LD (HL), C instruction. Remember that this will store the contents of the C register into the byte whose address is specified by the CONTENTS of HL: that is, it stores the character count at the start of the buffer.

An equivalent 8080 form avoiding relative jumps is shown in Fig 4, and again the code is based on the diagram structure. Remember — with the 8080 mnemonics, LXI loads a register PAIR and MVI loads a single register, thus MVI C,0 is placing zero into the C register, but LXI H, BUFFER$SPACE is placing the address BUFFER$SPACE into the HL register pair. Remember also that the letter 'M' represents the 8080 convention for an indirect address held in the HL register pair, thus LD (HL),A on the Z80 has an 8080 parallel instruction that is written as MOV M,A.

The 6502 version (Fig 5) performs the same essential functions but uses indexed addressing. We start by initialising the Y register to zero, then we use a loop to collect characters from the keyboard. If a character is not a carriage return, we increment Y (the character count) and store the character using STA (BUFFER$SPACE),Y. This is using indexed addressing to place the accumulator contents in the byte whose address is given by the base address (which the assembler calculates from your BUFFER$SPACE label), plus the offset held in the Y register.

To 'close' the buffer, we store the contents of the Y register at the start of the buffer. This is achieved by the instruction STY BUFFER$SPACE.

The three routines are all correct in that input data will be placed into the buffer as required, but we did say that there's a deliberate omission. What is it? In practice, the buffer can hold only 255 characters, so it's necessary to perform a check to see whether the buffer is full or not. Here's a couple of problems concerning this check.

Problem one

In every version we have shown, it's possible to add a single instruction to perform a suitable check. Think about the effect of incrementing the count as the buffer becomes full, and decide which flag will be affected. Use this flag to conditionally jump or branch out of the loop and perform the close buffer operation.

Sep 84 5(9) 2 of 4

Problem two
The test for possible buffer overflow should be indicated on the Warnier diagram. The mutually exclusive action subsets to be added are as follows:

BUFFER FULL (0,1 time) { SKIP AND EXIT ROUTINE

$\oplus$

$\overline{\text{BUFFER FULL}}$ (0,1 time) { STORE CHARACTER IN BUFFER

This pre-test alternation description can be superimposed on the existing Warnier diagram to reflect the change made to the code. When you have tackled problem one, try to redraw the Warnier diagram so that the coding changes are mirrored in the Warnier description.

Solutions
The first part should have been easy! The character count when it reaches 255 will increment to zero; thus buffer overflow can be detected by the setting of the zero flag. A simple but effective solution is to use a conditional branch or jump immediately after the instruction that increments the character count. By jumping to the CLOSE$BUFFER label, any over-sized entry will be safely ignored. The necessary changes are similar on all three processors, so we'll illustrate the idea with the Z80 form (Fig 6).

The addition to the Warnier diagram is shown in Fig 7. The extra operations occur, as should be expected, immediately after the INCREMENT COUNT statement.

Data movement

To move data from a buffer area to its 'final resting place' involves an understanding of some of the ways that blocks of data may be moved around in memory. To give some ideas of the approaches used, we'll look at typical coding. We are primarily interested in moving data from an area whose starting address is fixed (that is, our text buffer) to an area whose starting address will vary as data is added. To move a block of data we need to know three things:

a) Where the data is to be obtained from.
b) Where the data is to be transferred to.
c) The size of the block to be transferred.

In other words, we need a source pointer, a destination pointer, and a count of the number of bytes to be

```
GET$WORD:       LDY  #0                     ;Initialise count

BUILD$STRING:   JSR  INPUT$ROUTINE          ;System call
                CMP  #CARRIAGE$RETURN       ;Is it a CR?
                BEQ  CLOSE$BUFFER
                INY                         ;Increment count
                STA  BUFFER$SPACE,Y         ;Store character
                JMP  BUILD$STRING           ;Back for next character

CLOSE$BUFFER:   STY  BUFFER$SPACE           ;Store character count
                RTS                         ;Return from subroutine
```

Fig 5 Get-word 6502 version one

```
GET$WORD:       LD   C,0                    ;Initialise count
                LD   HL,BUFFER$SPACE        ;Start of buffer

BUILD$STRING:   CALL INPUT$ROUTINE          ;System call
                CP   CARRIAGE$RETURN        ;Is it a CR?
                JR   Z,CLOSE$BUFFER
                INC  C                      ;Increment count
                JR   Z,CLOSE$BUFFER         ;Z set + overflow
                INC  HL                     ;Increment pointer
                LD   (HL),A                 ;Store character
                JR   BUILD$STRING           ;Back for next character

CLOSE$BUFFER:   LD   HL,BUFFER$SPACE        ;Need start address again
                LD   (HL),C                 ;Store character count
                RET                         ;Return from subroutine
```

Fig 6 Get-word Z80 final version

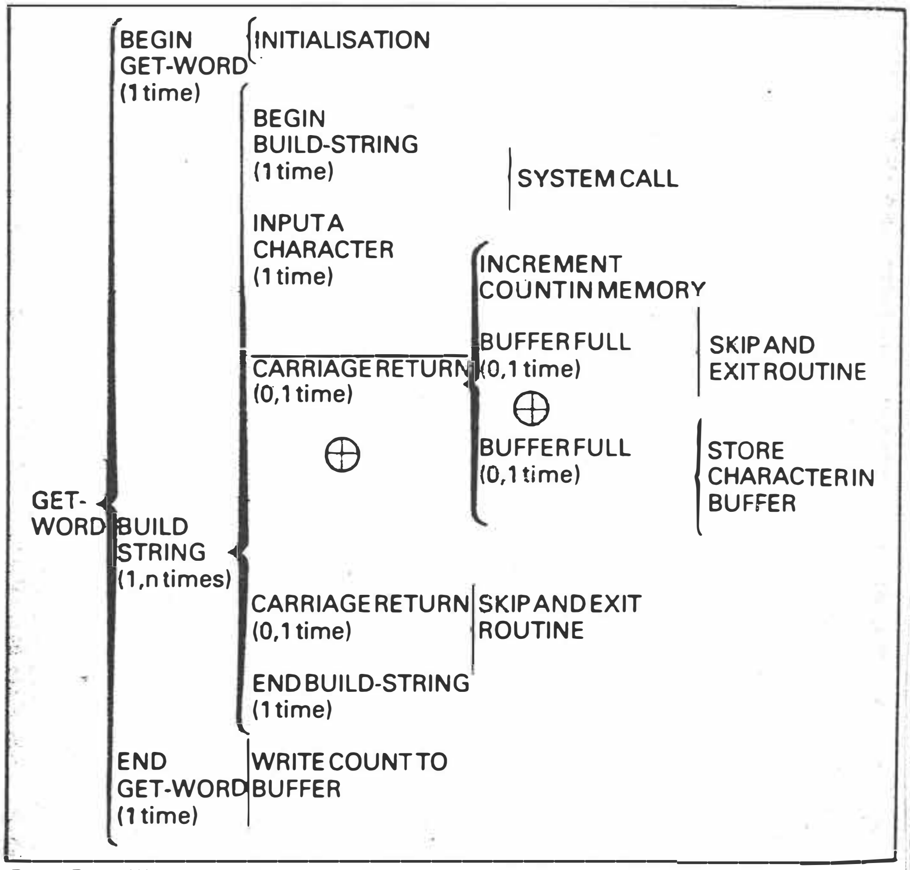


Fig 7 Final Warnier diagram

Sep 84 5(9) 3 of 4

```
ENTRY CONDITIONS:

HL  =  SOURCESTART ADDRESS
DE  =  DESTINATION START ADDRESS
C   =  NUMBER OF CHARACTERS TO BE TRANSFERRED

MOVE$BYTES: LD   A,(HL)            ;Get byte
            LD   (DE),A            ;Store byte
            INC  HL                ;Increment source pointer
            INC  DE                ;Increment destination pointer
            DEC  C                 ;Decrease count
            JR   NZ,MOVE$BYTES
            RET                    ;Return from subroutine
```

Fig 8 Move block Z80 version

```
ENTRY CONDITIONS:
HL  =  SOURCESTART ADDRESS
DE  =  DESTINATION START ADDRESS
C   =  NUMBER OF CHARACTERS TO BE TRANSFERRED

MOVE$BYTES: MOV  A,M               ;Get byte
            STAX D                 ;Store byte
            INX  H                 ;Increment source pointer
            INX  D                 ;Increment destination pointer
            DCR  C                 ;Decrease count
            JNZ  MOVE$BYTES
            RET                    ;Return from subroutine
```

Fig 9 Move block 8080 version

```
ENTRY CONDITIONS:

HL  =  SOURCESTARTADDRESS
DE  =  DESTINATIONSTARTADDRESS
BC  =  NUMBEROFCHARACTERS TOBE TRANSFERRED

MOVE$BYTES: LDIR                   ;Automated block move
            RET                    ;Return from subroutine
```

Fig 10 Automated move block Z80

```
ENTRY CONDITIONS:

Y = NUMBER OF BYTES TO BE TRANSFERRED

DESTINATION ADDRESS DEFINED IN ZERO PAGE MUST BE ONE BYTE
BELOW THE INTENDED DESINTATION ADDRESS

MOVE$BYTES: LDA  SOURCE$ADDRESS-1,Y        ;Get byte
            STA  (DESTINATION$ADDRESS),Y   ;Store byte
            DEY                            ;Decrease counter
            BNE  MOVE$BYTES
            RTS                            ;Return from subroutine
```

Fig 11 Move bytes 6502 version

transferred. On the 8080 and Z80, a byte of data may be transferred via the accumulator using HL as a source pointer and DE as a destination pointer. Thus the instructions needed on the 8080 are:

```
MOV A,M  ;Get byte
STAX D   ;Store byte
```

The equivalent Z80 instructions are written as:

```
LD A,(HL)  ;Get byte
LD (DE),A  ;Store byte
```

If a count of the number of bytes to be transferred is kept in the C register, a loop can be used to transfer up to 255 bytes from a source area to a destination area. A typical Z80 code is shown in Fig 8.

The 8080 version (Fig 9) incorporates the same ideas and should not prove too difficult to follow.

In the case of the Z80, a far more efficient alternative to the loops just described is available. The Z80 has incorporated in its instruction set some very powerful 'block move' instructions. In essence, the HL register pair is loaded as a source pointer, the DE pair as a destination pointer, and BC as a 16-bit byte counter. One such instruction using this pointer arrangement is the repeating block load with increment instruction whose mnemonic is LDIR. This instruction loads the contents of the byte addressed by HL into the location addressed by DE; HL and DE are then incremented and the BC pair decremented. If BC does not equal zero, the program counter is decreased by two and the instruction re-executed. The automated version of the Z80 loop shown earlier is given in Fig 10 for comparison.

On the 6502, we can move a specific byte from one address to another using the instructions:

```
LDA SOURCE$ADDRESS
STA DESTINATION$ADDRESS
```

This is all very well if only one byte is being moved and we know the addresses at the time we write the program, but when several bytes must be transferred, the indexed equivalent instructions may be used to move the Y'th byte of a page of data. The equivalent indexed forms are:

```
LDA SOURCE$ADDRESS,Y
STA DESTINATION$ADDRESS,Y
```

For the purpose of transferring data from a buffer such as we have described, we are particularly interested in moving data from a fixed base area (that is, the buffer area) to an area whose starting address may well vary (we could be transferring text into a dynamically changing 'string space' area). This being so, we will want to keep the destination address in two zero page locations and use indirect indexed addressing to define the destination address. The code that achieves this data movement will be of the form:

```
LDA SOURCE
    $ADDRESS,Y          ;Get byte
STA (DESTINATION
    $ADDRESS),Y         ;Store byte
```

One possible approach on the 6502 (Fig 11) is to use a backward counting loop to perform the above instructions Y times, decreasing the value of Y with each pass through the loop. As the loop that follows does not deal with the base address bytes themselves (that is, the case of Y=0), it's necessary to address the byte below the intended source start address. It's also important to ensure that the indirect pointer stored in the zero page is a pointer to the byte below the actual destination start address.

END

Sep 84 5(9) 4 of 4.

Sort At Input

by Tom Ithell

Sorting is the most written-about topic in software literature. Reams and reams have been written about chopping a few extra microseconds off a sort time.

When the data to be sorted is typed at a keyboard, the most obvious and frequently overlooked method is to sort at input. During the pause between pressing RETURN and the next data item, there's usually sufficient time to place the data item in a sorted array. The impressive aspect of this method of sorting is that a sorted output is immediately available after entering the last item.

The routines were written on a TRS-80 Model 1, although little modification is needed to run the routines in any dialect of Basic. Listing one is a sort of numbers into ascending order, listing two is a sort of strings into ascending order, and listings three and four show the changes needed to make the sort in descending order.

```
1 REM LISTING 1
10 REM NUMBER SORT ON INPUT
20 REM (C) T.A. ITHELL 1984
30 REM USEFUL FOR UPTO 200 NUMBERS
40 REM DELETE REM STATEMENTS FOR FASTEST OPERATION
100 CLS
109 REM SPECIFY READINGS
110 INPUT"STATE NUMBER OF ITEMS TO BE SORTED";NR
119 REM DIMENSION ARRAY
120 DIM ARRAY(NR+1)
129 REM INITIALISE ARRAY(0) WITH LARGE DUMMY NUMBER
130 ARRAY(0)=1000000000000000000
139 REM ZERO ARRAY
140 FORZ=1 TO NR+1
150 ARRAY(Z)=0
160 NEXTZ
169 REM NUMBER INPUT LOOP
170 FOR LOOP=1 TO NR
180 PRINTLOOP;:INPUT"STATE NUMBER";V
189 REM CHECK IF INPUT IS LESS THAN DATA ALREADY IN ARRAY
190 FOR CHECK=0 TO LOOP
200 IF V(=ARRAY(CHECK)THEN220
210 NEXT CHECK
218 REM MOVE ALL EXISTING SORTED NUMBERS FORWARD ONE ARRAY
219 REM ELEMENT TO CREATE SPACE FOR NEW NUMBER
220 FOR MOVE = LOOP TO CHECK STEP-1
230 ARRAY(MOVE+1)=ARRAY(MOVE)
240 NEXT MOVE
249 REM PUT NEW NUMBER INTO THE ARRAY
250 ARRAY(CHECK)=V
260 NEXT LOOP
269 REM PRINTOUT THE SORTED NUMBERS
270 FOR PR=0 TO NR-1
280 PRINT ARRAY(PR);" ";
290 NEXT
```

```
1 REM LISTING 2
10 REM STRING SORT ON INPUT
20 REM (C) T.A.ITHELL 1984
30 REM USEFUL FOR UPTO 100 STRING DATA ITEMS
40 REM DELETE REM STATEMENTS FOR FASTEST OPERATION
100 CLEAR2000:CLS
109 REM SPECIFY READINGS
110 INPUT"STATE NUMBER OF STRINGS TO BE SORTED";NR
119 REM DIMENSION ARRAY
120 DIM ARRAY$(NR+1)
129 REM INITIALISE ARRAY$(0) WITH LARGE DUMMY STRING
130 ARRAY$(0)="ZZZZZZZZZZZZZZZZZZZZ"
139 REM ZERO ARRAY
140 FORZ=1 TO NR+1
150 ARRAY$(Z)=""
160 NEXTZ
169 REM STRING INPUT LOOP
170 FOR LOOP=1 TO NR
180 PRINTLOOP;:INPUT"STATE STRING";V$
189 REM CHECK IF INPUT STRING IS LESS THAN DATA ALREADY IN ARRAY
190 FOR CHECK=0 TO LOOP
200 IF V$(=ARRAY$(CHECK)THEN220
210 NEXT CHECK
218 REM MOVE ALL EXISTING SORTED STRINGS FORWARD ONE ARRAY
219 REM ELEMENT TO CREATE SPACE FOR NEW STRING
220 FOR MOVE = LOOP TO CHECK STEP-1
230 ARRAY$(MOVE+1)=ARRAY$(MOVE)
240 NEXT MOVE
249 REM PUT NEW STRING INTO THE ARRAY
250 ARRAY$(CHECK)=V$
260 NEXT LOOP
269 REM PRINTOUT THE SORTED STRINGS
270 FOR PR=0 TO NR-1
280 PRINT ARRAY$(PR)
290 NEXT
```

```
1 REM LISTING 3
10 REM NUMBER SORT ON INPUT (DESCENDING ORDER)
!
DELETE LINES 129 AND 130
!
139 REM ZERO ARRAY
140 FORZ=0 TO NR+1
150 ARRAY(Z)=0
160 NEXTZ
!
189 REM CHECK IF INPUT IS GREATER THAN DATA ALREADY IN ARRAY
190 FOR CHECK=0 TO LOOP
200 IF V=)ARRAY(CHECK)THEN220
210 NEXT CHECK
```

```
1 REM LISTING 4
10 REM STRING SORT ON INPUT (DESCENDING ORDER)
!
DELETE LINES 129 AND 130
!
139 REM ZERO ARRAY
140 FORZ=0 TO NR+1
150 ARRAY$(Z)=""
160 NEXTZ
!
189 REM CHECK IF INPUT STRING IS GREATER THAN DATA ALREADY IN AR
RAY
190 FOR CHECK=0 TO LOOP
200 IF V$=)ARRAY$(CHECK)THEN220
210 NEXT CHECK
```

APC Jan 85 6(1)
p. 122-124.

The basic art

Mike Liardet, aided by 'The Art of Computer Programming', presents a beginner's guide to Basic programming through algorithms and information structures.

Computer programming is a craft. Given the raw ingredients of a programming language, a skilled programmer can blend them together into a fine working system by using his problem solving skill in conjunction with programming techniques that he has developed over a period of time. In an analogous fashion a traditional craftsman (a carpenter, for example) can transform a few pieces of wood into an exquisite piece of furniture by using different types of joints and various skills acquired over the years.

As with any craft the acquisition of skill comes partly with experience, but it can be more readily acquired by sound teaching and well-written text books. A valuable source of reference for anyone wanting to learn programming lies in a three-volume set of books by an American academic, Donald Knuth. These books are collectively entitled *The Art of Computer Programming**.

Knuth has planned seven volumes in the series, and has completed three volumes to date. Volume one introduces the basic concepts and defines what an 'algorithm' is, giving numerous examples; it also deals with 'information structures'. Volume two covers random numbers and arithmetic, and volume three deals with sorting and searching.

The books present the material as a pleasing blend of descriptions, formal presentation and set problems (and answers), and there are also interesting background histories and bibliographies. They have long been the computer science student's bible, but here they are presented for a new generation of apprentice programmers learning their craft outside the confines of academe. These books will be invaluable to anyone interested in what goes on 'under the bonnet' of computer systems.

Apart from the genuinely useful material, the books are also rich in a huge variety of algorithms that you always knew existed but were unable to find. One of my favourites is the algorithm to calculate when Easter falls (Fig 1). Easter is the first Sunday following the first full moon on or after 21 March. Did you know that this algorithm was devised by a Neapolitan astronomer in the sixteenth century? And that the only application of arithmetic in the Middle Ages was for determining Easter? Oh, yes — the volumes are a mine of information!

Knuth has invented an assembly language called MIX, which he uses to present the algorithms. We'll convert some of these MIX programs into the micro world's *lingua franca* — Basic.

Introduction

Volume one contains general introductory material, and begins by defining

```
2000 REM CALCULATE DATE OF EASTER FOR THE YEAR Y A.D.
2010 REM RETURNS N FOR DAY AND M FOR MONTH
2020 REM GET "GOLDEN NUMBER" (1 TO 19)...
2025 DEF FNREMN(A,B)=A-INT(A/B)*B:REM DEFINE REMAINDER FUNCTION
2030 G=FNREMN(Y,19)+1
2040 REM GET CENTURY C...
2050 C=INT(Y/100)+1
2060 REM LEAP YEAR AND LUNAR ORBIT CORRECTIONS...
2070 X =INT(3*C/4)-12:Z=INT((8*C+5)/25)-5
2080 REM FIND SUNDAY...
2090 D=INT(5*Y/4)-X-10
2100 REM CALC EPACT...
2110 E= FNREMN(11*G+20+Z-X,30):IF (E=25 AND G>11)OR E=24 THEN E=E+1
2120 REM CALC FULL MOON...
2130 N=44-E:IF N<21 THEN N=N+30
2140 REM ADVANCE N TO A SUNDAY...
2150 N=N+7-FNREMN(D+N,7)
2160 REM MARCH OR APRIL?
2170 M=3:IF N>31 THEN M=4:N=N-31
2180 RETURN
```

Fig 1 Easter algorithm

```
1000 REM EUCLID'S ALGORITHM
1010 REM RETURNS GREATEST COMMON DIVISOR OF M AND N
1020 REM ANSWER RETURNED IN N
1025 DEF FNREMN(A,B)=A-INT(A/B)*B:REM DEFINE REMAINDER FUNCTION
1030 R=FNREMN(M,N):REM CALC REMAINDER R FROM M/N
1040 IF R=0 THEN RETURN:REM N IS THE ANSWER IF ZERO REMAINDER
1050 M=N:N=R:GOTO 1030:REM OTHERWISE INTERCHANGE AND ROUND AGAIN
```

Fig 2 Euclid's algorithm

```
10000 REM INITIALIZE A DEQUE
10010 DIM X(100):DEQLEN=100:FRONT=1:BACK=1:RETURN
11000 REM ADD ITEM TO FRONT
11010 FRONT=FRONT-1:IF FRONT<1 THEN FRONT=DEQLEN
11020 IF FRONT=BACK THEN PRINT"OVERFLOW":STOP
11030 X(FRONT)=ITEM:RETURN
12000 REM ADD ITEM TO BACK
12010 X(BACK)=ITEM:BACK=BACK+1:IF BACK>DEQLEN THEN BACK=1
12020 IF FRONT=BACK THEN PRINT"OVERFLOW":STOP
12030 RETURN
13000 REM GET ITEM FROM FRONT
13010 IF FRONT=BACK THEN PRINT "UNDERFLOW":STOP
13020 ITEM=X(FRONT):FRONT=FRONT+1:IF FRONT>DEQLEN THEN FRONT=1
13030 RETURN
14000 REM GET ITEM FROM BACK
14010 IF FRONT=BACK THEN PRINT "UNDERFLOW":STOP
14020 BACK=BACK-1:IF BACK<1 THEN BACK=DEQLEN
14030 ITEM=X(BACK):RETURN
```

Fig 3 Deque processing

Feb 85 6(2) 1 of 4.

the word 'algorithm'. An algorithm is an unambiguous set of rules for performing a task which must be expressed in such a way that the algorithm always terminates. This condition is important. It is relatively easy to construct procedures that never terminate under some conditions — the 'infinite loop' that should be familiar to all programmers.

One of the earliest algorithms to be formally presented as such was Euclid's Algorithm to determine the greatest common divisor of two integers. (The greatest common divisor, or GCD, is the largest number that will divide both: for example, the GCD of 12 and 30 is 6.) The Basic variant is presented in Fig 2. The algorithm requires a 'remainder' or 'modulus' function; most versions of Basic don't have one, but the DEF FN facility can be used to create one:

DEF FNREMN (A,B) = A – INT (A/B) * B

Following the introduction to algorithms, Knuth outlines the basic mathematics needed to study some of the subsequent material. Unless you're mathematically inclined this is rather daunting, but fortunately isn't mandatory: the mathematics is needed for the theoretical study of the algorithms. The theory arises because it isn't sufficient to know that an algorithm will work; it's also important to know that it will work reasonably quickly. Determining information of this type can be very complex, and some of the material is devoted to it. However, if you're non-mathematical, or in a hurry, or both, you can safely skip this analysis and read the conclusions, not the proofs.

The next section describes the MIX assembly language, devised by Knuth and used in the description of some of the algorithms. (Descriptions are also given in a more familiar English-cum-programming language.) MIX is roughly equivalent to a typical 8-bit or 16-bit assembler available for most micros, but being a Knuth invention it doesn't commit the book to any one computer. Among other things the code for a MIX simulator is given, so if you're really keen you can get MIX up and running on your own machine and use it to work through some of the exercises. This is a good way to learn assembler programming.

Information structures

Following the introductions, volume one gets down to business with a comprehensive guide to information structuring. Most interesting programming tasks, especially non-numerical work, demand some skill at structuring data. In fact, some programming languages implement many of the facilities described by Knuth. Artificial Intelligence languages, such as Lisp, Prolog and Logo, are particularly rich in these features, but if you're working with other languages, Basic for example, then Knuth provides a thorough grounding for building up these facilities from scratch.

The most elementary structure is the sequentially allocated list, simply represented in Basic as a one-dimensional array: for example, DIM X(1000). This structure is quite adequate for tasks where the data to be stored is fixed during initialisation and left alone thereafter, but it can be cumbersome for dynamic structures, where elements may be added and deleted 'at random' throughout program execution. In order to insert an element at some point, all the elements after it must be shuffled along to make room, which can be very inefficient if the list has more than a few elements. Likewise, a deletion necessitates a shuffle in the other direction.

There's a special case where this arrangement *can* work efficiently, and this is when all insertions and deletions take place only at the ends of the list; this is known as a 'deque'. The deque concept includes two further, even more special, cases — the 'stack' and the 'queue'. Stacks add or delete data from one end, and queues add data at one end and remove it from the other. Both are very widely used — queues for buffering characters prior to processing, and stacks for managing computations on recursive structures.

The code for the four basic deque operations, plus initialisation, is given in Fig 3. Two variables (FRONT) and BACK) are used to mark the position of the ends of the deque. This should be obvious but some care is needed to check for 'overflow', when no further storage is available to accommodate an insertion, and 'underflow', when no data is there to be deleted.

It's also convenient to use the elements in the array as if they were arranged in a circle, so that the third follows the second which follows the first, but the first also follows the last. Queuing operations propel the deque through memory and without this trick would quickly fail, even if the deque were comparatively empty. Note that the BACK pointer marks the next position for an addition to the back of the deque, not the position of the last element. This wastes one location in the array, in the sense that an overflow will occur when one location is still free, but without doing this it is much more difficult to differentiate between an empty deque and an overflowed one.

Knuth devotes a lot of attention to the issue of storage management, as good storage management minimises prob-

```
 LIST
  !
  !
  V
+------+---+    +------+---+    +------+---+    +------+---+    +------+---+
! 101  ! ---> ! 102  ! ---> ! 103  ! ---> ! 104  ! ---> ! 105  ! 0!
+------+---+    +------+---+    +------+---+    +------+---+    +------+---+
```

Fig 4 Diagrammatic linked list

```
 LIST
  !
  !
  V
+------+---+    +------+---+    +------+---+    +------+---+    +------+---+
! 101  ! ---> ! 102  ! --+    ! 103  ! ---> ! 104  ! ---> ! 105  ! 0!
+------+---+    +------+---+!   +------+---+    +------+---+    +------+---+
                            !
                            +-------------------+
```

Fig 5 Insertion and deletion by manipulating pointers

```
 LIST
  !
  !
  V
+------+---+    +------+---+    +------+---+    +------+---+    +------+---+
! 101  ! ---> ! 102  ! ---+  ! 103  ! ---> ! 104  ! ---    ! 105  ! 0!
+------+---+    +------+---+ !  +------+---+    +------+---+    +------+---+
                             V        ^
                    +------+---+      !
                    !102.5! ---+
                    +------+---+
```

Fig 6 Adding a new node to the list

Feb 85 6(2) 2 of 4.

lems with storage overflow. For example, if there are several stacks, queues or deques used by a program which are all initialised with fixed capacity, the program fails as soon as one overflows even though many of the others are almost empty. This unsatisfactory state of affairs can be improved by arranging for all the available storage to be pooled, then allocated in small chunks as it's needed. If a deque overflows, a larger storage area can be requested from the pool, the data copied across, and the old storage area returned to the pool of free storage. Storage management is also useful for handling any other information structures, such as linked lists. The linked list solves the insertion-deletion problem of the sequentially allocated list. Each item in the list is stored along with a pointer to the next, which therefore need not be adjacent in memory. For example, a list of the numbers 101 to 105 can be represented in memory as follows:

Location	**Contents**	
	Data and Pointer	
1 & 2	101	7
3 & 4	not used	
5 & 6	103	11
7 & 8	102	5
9 & 10	105	0
11 & 12	104	9
13 onwards	not used	

In this example, both data and pointer each require one storage location, but it's possible to have lists where this is not the case and even where the amount of data varies between the different 'nodes'. The pointer following 105 is 0. As 0 is an impossible location (in this example), this indicates the end of the list. A linked list can be drawn diagrammatically as in Fig 4.

Insertion and deletion in a linked list is handled by manipulating the pointers: for example, deleting the node with 103 is achieved by changing 102's pointer (Fig 5).

Ideally, the node at 103 should be handed back to the pool of free storage so that its storage area can be re-used later: for example, if you wanted to add a new node to the list (Fig 6).

Apart from their use in representing live data, linked lists also form the basis of many storage management algorithms. At any given moment in the program's execution, the free storage will be in several isolated fragments. A single variable indicates the location of one — any one will do. This contains a pointer to another, and so on. Unless all allocations and deallocations are for a fixed size, the size of each will need to be recorded. As long as the node is large enough this can be stored with the pointer, thus each free node may start with a size value, then a pointer, and then the remaining free space. Assuming the pointer and size value each consume one location, a typical free list is shown in Fig 7.

'... if you're really keen you can get MIX up and running on your own machine and use it to work through some of the exercises. This is a good way to learn assembler programming.'

The code to manage such a storage list is given in Fig 8, and demonstrates that there's nothing difficult about storage management. The free storage area is the array X(), which is initialised as just two free blocks; the first of length 2, with the second immediately following it and occupying the rest of the array. The first block is never allocated, but is kept solely for its pointer to the next free block. If this pointer were held in a variable, FREELIST, for example, then changes to the first block would need to be coded as a special case since they would alter the value of FREELIST and not a pointer in X(). In this example, all storage requests must be for an even number of locations as this will guarantee that no free blocks of length 1 are created. (A block of length 1 cannot contain a pointer *and* a length value.) The deallocation routine can be made a lot more effective by arranging for adjacent free blocks to be merged together. As it stands, storage will become more and more fragmented until the free storage is just a long chain of tiny blocks.

Knuth has much more to say on storage management including garbage collection, where it's unnecessary to explicitly free a block when it's no longer needed — the system can work this out for itself. There are also many alternative algorithms for maintaining freelists, each with pros and cons which are discussed at length.

Another major type of information structure is the 'tree'. A tree is more complex than a linked list in that each node contains several pointers, not just one. The pointers are to the 'children' of the node (the jargon for computer trees borrows heavily from that of family trees); these children in turn may point

```
FIRSTFREE
   !
   !
   v
+-----+-----+----------------------+
! 12  !  !  ! 10 unused locations  !
+-----+--!--+----------------------+
         !
   +-----+
   !
   v
+-----+-----+----------------------+
! 24  !  !  ! 22 unused locations  !
+-----+--!--+----------------------+
         !
  etc....etc
   !
   v
+-----+-----+----------------------+
! 18  !  0  ! 16 unused locations  !
+-----+-----+----------------------+
```

Fig 7 Typical free list

```
20000 REM INITIALIZE FREE STORAGE AREA
20010 DIM X(1000)
20020 X(1)=2:X(2)=3:X(3)=9998:X(4)=0
20030 RETURN
21000 REM RESERVE N UNITS OF STORAGE. ADDRESS OF BLOCK IN LOCN
21010 Q=1
21020 P=X(Q+1):IF P=0 THEN PRINT"STORAGE OVERFLOW":STOP
21030 IF X(P)<N THEN Q=P:GOTO 21020
21040 K=X(P)-N:IF K=0 THEN X(Q+1)=X(P+1):GOTO 21060
21050 X(P)=K
21060 LOCN=P+K:RETURN
22000 REM RETURN N UNITS OF STORAGE AT LOCN
22010 X(LOCN)=N:X(LOCN+1)=X(2):X(2)=LOCN:RETURN
```

Fig 8 Storage management routines

Feb 85 6(2) 3 of 4

to grandchildren, and so on. It isn't usually desirable for each node to have a different number of pointers, depending on the number of children, so frequently just two pointers are used: one to the first child; and another to the next sibling of the node. For example, the tree structure:

```
        A
       / \
      /   \
     B     C
     !    /!\
     !   / ! \
     D  E  F  G
```

is represented with two pointer nodes as:

```
          A
         /
        /
       B>>>>>>>>C
      /        /
     /        /
    D        E>>F>>G
```

(where '>> . .' denotes sibling pointers). With this structure it is only slightly more difficult to access, say, the Nth child of a node, than it would be with multiple pointers. Note that A, C, D and G have no 'younger' siblings, so their pointers are simply null. Likewise D, E, F and G have no offspring.

Tree structures can be very useful for working with mathematical expressions, where the tree structure exactly represents the order of evaluation: for example, 3 * LN(X + 1) – A * X^2 is represented as a tree:

```
            -
           / \
          /   \
         *     *
        / \   / \
       /   \ /   \
      3    LN A   ^
           !     / \
           !    /   \
           +   X     2
          / \
         /   \
        X     1
```

Knuth develops all the algorithms necessary for symbolically differentiating such a tree. The answer is generated as another tree structure, and issues such as copying tree structures, and ordering the nodes for evaluation, are all dealt with along the way.

With the above representation, it's not readily possible to determine the parent of a node as there are no pointers back to it. In tree processing, it's usual to maintain a stack of the parents *en route* to the current node — the earlier work on stacks and queues is of value here. It's also possible to use the null pointers of childless or youngest-sibling nodes to point back; this is called a threaded tree. It has the advantage that no stack is needed, so it's impossible for it to overflow. To determine whether your Basic interpreter uses threaded or unthreaded trees for evaluation of expressions, type:

PRINT ((((..(((3 + 4)))..))))

for ever more pairs of brackets. If it eventually gives a memory error, this is a strong indication that it's using a stack for handling the evaluation. Microsoft's Basic finally runs out of space with 72 pairs of brackets — not a serious limitation!

The last major topic to be covered under Information Structures is the 'array', which is represented in Basic by the multiply dimensioned use of DIM. For example, DIM A(3,3,3) defines a 3D array with 27 elements (or 64 if the lower bounds are 0 and not 1).

Representing arrays in this fashion can be highly inefficient if the contents of the array are sparse. Some arrays are triangular, with all zeroes above the diagonal, or diagonal with all zeroes except the diagonal, and so on. Knuth suggests a linked allocation method for these circumstances. Each non-zero element in the array is represented by a node containing the value, its row and column numbers, and pointers to the next (non-zero) node above it and to the left of it. Zero elements are not there, so consume no extra storage. In addition, each row and column starts with a dummy node, not an array element, but eases the processing of empty rows and columns.

A further refinement is for the linkages to be 'circular'. This means that the pointer in the last node of a sequence is not null, but points to the first node. With this scheme of pointing, the notion of first and last disappears. Such an array, containing only three non-zero elements, is shown in Fig 9.

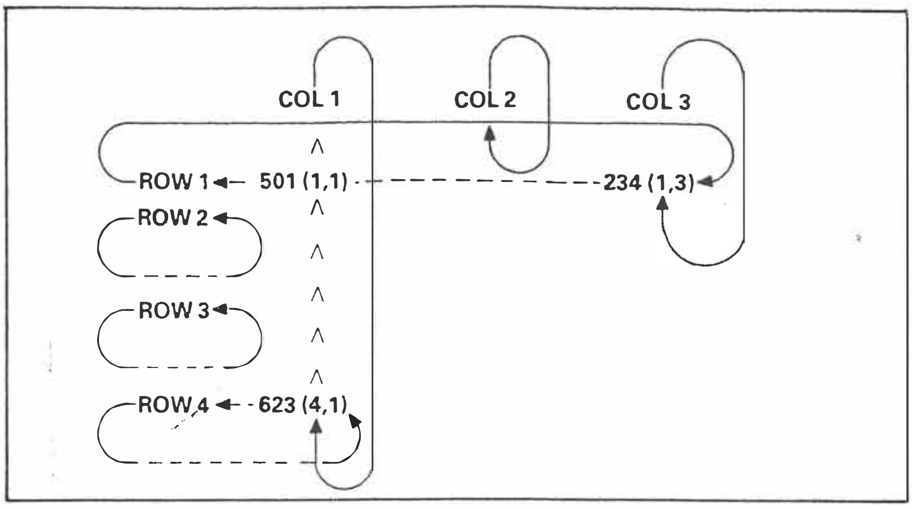


Fig 9 Circular array containing three non-zero elements

With this type of structure, great storage savings can be made with large sparse arrays, and the access time for any given element need not be excessive.

As the array is sparse there should be only a few elements on each list. This means that it should be comparatively fast to find any element.

It's even faster if the array is being scanned in some systematic fashion, as is the case with most numerical algorithms. It's also worth noting that this type of representation allows for new rows and columns to be inserted with minimal changes to the structure, or data moving.

Conclusion

We have taken a look at the first of Knuth's *Art of Computer Programming* volumes, and introduced most of the major topics dealt with in the book. I hope the reader's appetite for improving his programming techniques is sufficiently whetted, and strongly recommend this book as an instructor and reference manual.

**References*

The Art of Computer Programming *by Donald E Knuth; Addison-Wesley Publishing Company.*

Volume 1 Fundamental Algorithms.

Volume 2 Seminumerical Algorithms.

Volume 3 Sorting and Searching.

END

Feb 85 6(2) 4 of 4.

Pick a number

For those who want to learn about number-crunching and arithmetic on their micro, Donald Knuth's second volume in his trilogy may be the answer, as Mike Liardet explains.

Seminumerical Algorithms is not the most welcoming title for a book. But when the author is Donald Knuth and the volume in question is the second in his *The Art of Programming* trilogy, then any reservations are worth overcoming.

The title of this second volume is in fact a little strange, but Knuth justifies the *'Semi'* prefix on the grounds that the book also concerns itself with the tactics of implementing efficient algorithms for numerical work: it deals with random numbers, and arithmetic. It does not get heavily involved in the specialist field of numerical analysis, although many of the topics would be of interest to numerical analysis workers.

> *'Unlike the random number sequences generated by software, Tattslotto is more truly random in that it generates numbers on the basis of measuring random physical phenomena.'*

Random numbers

Random numbers — that is, numbers 'chosen at random' — are useful for simulation, modelling, software validation, games playing and a variety of other applications. Perhaps the best-known random number generator is the Tattslotto barrel. Unlike the random number sequences generated by software, the barrel is more truly random in that it generates numbers on the basis of random physical phenomena. Strictly speaking, pure software can only generate 'pseudo-random' numbers: if you know or can guess the underlying algorithm, then the sequence will appear completely non-random, to you at least, since you will be able to predict the entire sequence. Thus the randomness is only an illusion for the uninitiated.

John von Neumann, the father of the modern electronic computer, was the first to propose a simple algorithm for generating pseudo-random (from now on 'random') numbers: to generate the next random number in a sequence, square the previous one and pull out the middle digits as the next random number. The following Basic code generates four-digit random numbers:

```
DEF FNCMOD (U,V) = U-INT (U/V)*V
  :REM REMAINDER FUNCTION
DEF FNCVN (X) = FNCMOD
  INT (X*X/100), 1000)
```

Given some starting value for X, say 9876, then successively evaluating the expression X = FNCVN(X) will generate 5353 (middle four digits of 9876 x 9876 = 97535376), 6546 (middle four of 5353 x 5353 = 28654609), and so on. It should be obvious that sooner or later our *random* sequence will repeat itself. This happens immediately it generates a number previously generated. At best this could happen after 10,000 iterations, when every number from 0 to 9999 had occurred precisely once.

But, in practice, it happens much sooner. Starting from 9876 the generator quickly gets locked into a 'cycle' of four values: 5600, 4600, 3600, 9600, 5600, and so on. Starting from other values: if 0 is generated, then it continues to produce just 0 thereafter — hardly random behaviour!

The solution to this difficulty is to:

(a) use a better random number generator; and

(b) ensure that it works with numbers which are larger than you really need (you can always truncate unwanted digits from a large number).

Knuth introduces his own early effort to improve upon von Neumann's method which I will not discuss here because it is too complicated; however, in essence, it iterates a random number of times through several lines of arithmetic, starting at a random place for each iteration. Superficially this appears to be fairly promising but Knuth quickly discovered that it started repeating fairly quickly and was little better than von Neumann's method.

In fact very effective but simple and comprehensive random number generators can be written using the 'linear congruential method'. This is frequently used as the basis for the RND() function, familiar to most users of Basic. Given the previous random number, x, in a sequence, the next random number is calculated as:

(ax + c) mod m

where a, c and m are some carefully chosen constants. The term 'linear congruential' describes this expression — 'ax + c' is linear (that is, a straight line graph) in x, and congruential arithmetic is that which uses the mod function. Some versions of Basic are reputed to have fairly poor random number generators and this is probably because of a bad choice for the three constants. If your Basic is in this category, then you can easily use your own random number generator with:

```
DEF FNCLC(X) = FNCMOD (A*X+C,M)
```

The numbers generated by this method all lie in the range 0 to m–1 (the 'mod', or remainder, function guarantees this), so at best the sequence will repeat after m numbers have been generated. Choosing a large value for m can help, but bad values of a and c can also produce poor results. For example, a=1, c=2 produces 0, 2, 4, 6, and so on, from a starting value of 0. Much of Knuth's description of the method is devoted to the choice of good values for a, c and m.

We have already noted that m should be large, even if the required range of the random numbers is small. For example, for coin-tossing we could try m=2, then conveniently each random number would be either 0 (for heads) or 1 (for tails). However, this would, at best, produce the repeating sequence 0, 1, 0, 1, ... Choosing a high value for m would be far more satisfactory, then heads or tails could be denoted by the parity of the number, but the number itself would be retained as the input for the next random number.

When working in assembler it is simpler to code and faster to execute if m is

Mar 85 6(3) 1 of 5.

```
20000 REM CHI-SQUARED TEST FOR MSBASIC RND() FUNCTION
20010 DIM NUMCOUNT(50):REM HOLDS NUMBER OF OCCURENCES OF EACH NUMBER
20020 REM GENERATE 1050 RANDOM NUMBERS IN RANGE 0 TO 50...
20030 FOR I=1 TO 1050
20040 RAND=INT(RND(1)*51):NUMCOUNT(RAND)=NUMCOUNT(RAND)+1
20050 NEXT I
20060 REM NOW CALCULATE VARIANCE V...
20070 V=0
20080 FOR I=0 TO 50
20090 V=V+(NUMCOUNT(I)-20)^2/20
20100 REM (20 IS EXPECTED NUMBER OF OCCURENCES)
20110 NEXT I
20120 PRINT"V =";V
```

Fig 1 Calculation of variance for chi-squared test

restricted to a power of 2, especially the byte or word-size of the computer (this is irrelevant in Basic). For most values of m, mod m can only be calculated by using division, but, for example, if m = 2^8 = 256, then mod m for any number is produced by zeroing everything except the least significant byte of the number: for example, (in hexadecimal) 4321 mod 100 = 0021, or 6789 mod 100 = 0089. Knuth also shows an easy method for calculating mod 101, which is given here for those who are well versed in hexadecimal arithmetic. For 4321 mod 101:

Complement 4321: BCDE
Subtract low byte from high:
  00BC − 00DE = FFDE
If result is negative (which it is
  because BC < DE)
    then add 101:
      FFDE + 0101 = 00DF
And that's the answer!

Obviously these techniques can be readily extended for m = (hexadecimal) 10000, 10001, 1000000. The advantage of using 101 instead of 100, in a random number generator, is that with the latter the right-hand digits are much less random than the left.

Clearly a linear congruential random number generator must repeat after m numbers have been generated, but is it possible to choose values for a and c, such that m different values are always generated before repetition? The answer is yes. Trying a=c=1 always does this, although it is rather a predictable random sequence. But there are generally more effective values that can be chosen, as long as the following rules are observed:

**none of c's prime factors can be prime factors of m;*
**a-1 must be a multiple of every prime factor of m; and*
**a-1 must be a multiple of 4 if m is a multiple of 4.*

(The prime factors of a number are the prime numbers — numbers only divisible by themselves and one — which must be multiplied together to produce the number. For example, the prime factors of 100 are 2 and 5, since 2 and 5 are prime, and 2 x 2 x 5 x 5 = 100.) If m = 2100 (= 2 x 2 x 3 x 5 x 5 x 7), then c could be any number without these factors: 11, 13.. 121, and so on. a-1 must be a multiple of each of 2, 3, 5, 7, and also a multiple of 4 (because m is). Therefore, one possible value for a-1 would be 2 x 2 x 3 x 5 x 7 = 420, meaning a = 421.

All random numbers generators need to be 'started off' with some initial random, or 'seed' as it is termed. Generally, during program development it is expedient to assign some arbitrary constant as the start-up value. This means that the same sequence will be used every time the program is run, and any bugs in the software will be repeatable, and easy to correct.

Once the program is working correctly, it is undesirable to use the same sequence every time — if it's a card game you do not always want to be dealt the same cards! A useful way to create the seed is to access the date and time, if available, or to loop and increment the seed value when waiting for keyboard input, or restart with the last random number used at the end of the previous session. In any of these cases the random number generator should get off to a different start every time.

Knuth outlines many other possible algorithms for random number generation, involving slightly more complex calculations. An obvious extension to the linear congruential method is the quadratic congruential:

```
DEF FNCQC(X) =FNCMOD
  (A*X^2+B*X+C,M)
```

and there are many interesting generators that use two or more previous values to generate the next random number, including the simple, but poor, Fibonacci sequence:

```
DEF FNCFIB(X,XPREV) =
  FNCMOD(X+XPREV,M)
```

(This must be used by:
```
XNEW=FNCFIB(X,XPREV):XPREV=X:
  X=XNEW)
```

Of course, much of the foregoing provides a great deal of fertile ground for creating random number generators, but neatly skirts around methods for evaluating how good they are. For example, we have considered possible and convenient candidates for a, c and m in a linear congruential generator, but m=2100, a=421 and c= 11 (all mentioned above) generate random numbers that are alternately odd and even. Much of Knuth's treatise on random number generators is dedicated to tests, which should trap the unsatisfactory generators, and pass the good ones.

One of the simplest tests is known as the chi-squared test. This is a test used widely by statisticians, but in this context we can use it to gauge the evenness of distribution of a random number generator. If we use a generator a thousand times to generate numbers in the range 0 to 50, we would expect each number to turn up roughly 20 times, but even with truly random numbers we would, on average, expect a few oddities: perhaps one or two numbers would only turn up a few times.

With the chi-squared test, we can measure this evenness of distribution (using the program in Fig 1), by calculating the variance, V. This value can be looked up in a table (see Fig 2) which indicates what percentage of the time it would be expected. When I ran this program in Microsoft Basic, V was 60.1 on the first run. Examining the table shows that 56.33<60.1<67.5. We can expect V to be greater than 56.1 in 25 per cent of cases, so this run of the random number generator produced a fairly 'average' distribution, which is what we want. (Very low values of V are 'too good to be true', and very high values indicate obvious biases.)

There are many other tests that can be

mar 85 6(3) 2 of 5.

p=1%	p=5%	p=25%	p=50%	p=75%	p=95%	p=99%
29.71	34.76	42.94	49.33	56.33	67.50	76.15

Fig 2 Chi-square values for distribution on 51 random numbers

applied to random number generators, with intriguing names like the poker test, spectral test, and so on, and Knuth outlines them all in detail. To get the seal of approval, a random number generator should pass all of them. But what about those in a hurry, who need a highly recommended generator *on a plate?* Knuth outlines his own recommendation for such people at the end of this chapter (Fig 3). As presented by Knuth, the generator produces random numbers from Fortran routines in the range 0 to 999999999. I have translated it to Basic, using a floating point array to hold integer values in the range 0 to 9999999. (Basic integer arrays only handle numbers up to 32767 and Microsoft Basic floating point is only accurate to seven digits.)

Once an initial sequence of numbers has been set up, this random number generator generates the next random number from the difference between the random numbers given 55 and and 24 times previously. The result is taken mod 10000000. Most of the complications in the software arise from the book-keeping necessary to maintain 55 previous values in the sequence. The use of '55' and '24' are highly significant, and were definitely not picked at random. With these values, the random number generator will not start repeating for several millennia, even at computer speed! Knuth gives a number of other pairs of values that also work very well.

Arithmetic

The chapter on arithmetic is primarily concerned with the basic operations of addition, subtraction, multiplication and division. Subsequently, it introduces a number of related topics such as factorisation, exponentiation and polynomials. Users of high-level languages may think that much of this is of little interest, since the algorithms are already written for them. This attitude is a little

```
25000 REM KNUTH'S RANDOM NUMBER GENERATOR
25010 DIM RAND(55):REM GENERATES 55 NUMBERS AT A GO
25015 DEF FNCMOD(U,V)=U-INT(U/V)*V:REM BASIC'S MISSING MOD FUNCTION
25020 SEED=1234567!:REM VALUE TO GET IT STARTED
25025 GOSUB 30000:REM INITIALIZE
25030 PRINT"HERE'S A HUNDRED RANDOM NUMBERS..."
25035 FOR I=1 TO 100
25040 GOSUB 32000:PRINT X
25050 NEXT I
25060 STOP
30000 REM INITIALIZATION RAND() ARRAY STARTING WITH SEED VALUE
30010 RAND(55)=SEED:J=SEED:K=1
30020 FOR I= 1 TO 54
30030 II=FNCMOD(21*I,55)
30040 RAND(II)=K
30050 K=J-K:IF K<0 THEN K=K+10000000£
30060 J=RAND(II)
30070 NEXT I
30080 REM NOW WARM UP THE GENERATOR...
30090 GOSUB 31000:GOSUB 31000:GOSUB 31000:RETURN
31000 REM RESET RAND() ARRAY WITH NEW VALUES IN RANGE 0 TO 9999999
31010 FOR I=1 TO 24
31020 J=RAND(I)-RAND(I+31)
31030 IF J<0 THEN J=J+10000000£
31040 RAND(I)=J
31050 NEXT I
31060 FOR I=25 TO 55
31070 J=RAND(I)-RAND(I-24)
31080 IF J<0 THEN J=J+10000000£
31090 RAND(I)=J
31100 NEXT I
31110 NEXRND=1
31120 RETURN
32000 REM AFTER INITIALIZATION, RETURNS RANDOM NUMBER IN RANGE 0 TO 9999

32010 IF NEXRND>55 THEN GOSUB 31000
32020 X=RAND(NEXRND):NEXRND=NEXRND+1
32030 RETURN
```

Fig 3 Knuth's recommended random number generator

mar 85 6(3) 3 of 5.

short-sighted, as a good understanding of these underlying algorithms should enable the user to program with maximum precision!

The simplest form of computer arithmetic is fixed-point arithmetic. In fixed-point arithmetic the amount of storage space for every number is the same, and the decimal point is always understood to be in the same place. The most usual convention is for it to be after the last (least significant) digit; and in this case the computer is performing integer arithmetic. The advantage of integer arithmetic is that it is fast, and excepting loss of remainders in division, completely accurate for the four main arithmetic operations. The disadvantage is that it cannot represent very large magnitiude numbers, at least not without allocating a lot of storage.

Fixed-point software for 8-bit micros usually allocates two consecutive bytes, totalling 16 bits of storage for each integer. Some software or 'double precision' options may offer more. As each bit (=' binarydigit') can hold just two values (0 or 1), 16 bits together allow 2 x 2 x... x 2 = 2^16 = 65536 different integer values to be represented.

Generally it is undesirable that only positive numbers be accommodated, and Knuth describes different methods for handling negative numbers. The most popular is the 'two's complement', where the most-significant bit (that is, the leftmost when writing the number on paper) is always 1 for negative numbers:

1000 0000 0000 0000 = −32768
1111 1111 1111 1111 = −1
0000 0000 0000 0000 = 0
0000 0000 0000 0001 = 1
0111 1111 1111 1111 = 32767

This representation is somewhat analogous to a counter on a cassette recorder. If you set it to zero in the middle of a tape, and then rewind, it progresses back through 999, 998, and so on. One advantage of it is that no special action need be taken for adding negative numbers: the computer's normal binary add instruction should work. Negating a number is also fairly easy: just 'complement' it (a single computer instruction that changes all 1s to 0s and vice versa) and add one. For example to negate 1:

Complement 0000 0000 0000 0001 =
1111 1111 1111 1110
And add 1: 1111 1111 1111 1111.

With 16-bit two's complement arithmetic there is no facility for representing numbers less than −32768 or greater than 32767, and correctly implemented software will generate an 'overflow' error if a calculation oversteps the mark. If you try this in Basic (for example, PRINT 32767 + 1) you may be surprised to see that the correct answer is displayed instead of an error, (but you can force the error by typing LET X% = 32767 + 1). Many versions of Basic avoid integer overflow by converting the result to floating point.

Floating point numbers

The representation of floating point numbers in the computer is analogous to the scientific notation, where very large or small magnitude numbers are represented by a fraction and exponent part. For example, in scientific notation Planck's constant would be written as 1.0545 x 10^−27. (Basic uses a minor variant of this notation: 1.0545E−27.) The fractional part is 1.0545, and the exponent is −27. This number could otherwise be written (with spaces added for readability):

0.00000 00000 00000 00000
00000 010545

Notice that this number is the original fraction 1.0545 with the decimal point shifted 27 places to the left. In scientific notation the convention is to place the decimal point of the fraction only after the first digit. For example, 105.45 x 10^−29 and .010545 x 10^−25 also equal Planck's constant, but not in the normal representation. This principle also holds for most floating point software.

Typical floating point software on an 8-bit micro represents a number by using at least four consecutive locations: the first is used to hold the exponent of the number, and the remainder are used for the fractional part. It is obviously desirable to accommodate both negative and positive exponents, so the positive integer value stored in the exponent must have some 'excess' quantity subtracted to reveal its true value. A single byte could hold any value from 0 to 255, which, if the excess were 128, would allow the exponent to range between −128 and 127. The decimal point for the fractional part is usually to the left of the most significant digit, and the normalisation requirements say that this (binary) digit should be 1. To avoid confusion between normalised and unnormalised numbers, the position occupied by this bit can be used to store the sign of the number. The number zero is uniquely represented by all bytes including the exponent, being zero.

All floating point operations, even addition, can introduce inaccuracies into the results. This is because the fractional part of the result can easily require more space than is allocated for it to be represented with complete accuracy, and it must be 'rounded' to fit in. These inaccuracies can be lessened by arranging for double precision storage during the calculation, but the returned result must be returned at normal size.

It is possible to gain some intuition into the workings of floating point software, by working with scientific notation, and restricting the number of digits in both the fractional and exponent parts. For example, with just four digits for the fraction and one for the exponent, consider the following addition and multiplication:

(1) Add 8.765E−2 to 9.998E1
Adjust 8.765E−2 to have exponent E1: 0.008765E1
Add 0.008765E1 to 9.998E1 =
10.006775E1
Normalise the result: 1.0006765E2
And round to four digits: 1.001E2
(2) Multiply 3.111E7 by 9.000E−4
Add exponents: 7 + −4 = 3
Multiply fractions: 3.111 x 9.000 =
27.999
So the product is: 27.999E3
Normalise it: 2.7999E4
And round to four digits: 2.800E4

As with fixed point arithmetic, it's possible to have an overflow condition in a floating point operation. This occurs if

Puzzle solution

Of course the two expressions are equal. The presence of rounding error can result in minor differences when they are evaluated on a computer, but to obtain such a gross difference we have to arrange for one expression to underflow and the other not to.

This solution is specific to Microsoft Basic, but the principles should hold for any language that does not trap arithmetic underflow as an error. There are many possible values that will work, but I have checked the following on both CP/M and MS-DOS versions of Microsoft Basic:

A = 1E-30
B = 2.938735E-9
C = 1.701412E38
D = 2000

Evaluating AxB in the first expression causes an underflow, so the whole expression evaluates to zero. The second expression does not underflow, and returns a correct result, approximately 1000. (You can verify this by hand if you know that 2.938735 x 1.701412 = 5). By choosing ever larger values of D you can make the discrepancy even worse!

Mar 85 6(3) 4 of 5.

the exponent part gets too great. This would have happened in the above multiplication example if the second number had been 9.000E4 and not 9.000E-4: the product's exponent would then be two digits, and one more than we allowed for. In practice, real floating point software allows larger exponents than this, typically accommodating numbers as big as 10 38.

With floating point arithmetic it's also possible to have 'underflow' This occurs if the exponent part gets less than the lowest negative value permissible — that is, when the number is very close to zero. Computer users pay far less attention to underflow than they do to overflow or rounding, but Knuth rightly points out that its effects are just as insidious. In Microsoft Basic any number smaller than 2.938735E-39 underflows to zero. This may not appear to be worth worrying about, and indeed many language implementors, Microsoft included, do not give an error message for underflow. But underflow can cause a gross calculation error, with answers inaccurate by thousands, as you will gather if you can solve the following puzzle using your Basic interpreter:

Assign values to A, B, C and D such that ((AxB) xC) D differs from (A x (B xC)) xD by a thousand (solution in box at the end of article).

Conclusion

Knuth's section on arithmetic covers a great deal more than I have been able to mention here. For example, there are other less commonly used ways of representing numbers in the computer, and efficient algorithms for multiplication, and more besides — a veritable treasure trove for number-crunchers everywhere!

Readers after more treasure should look at last month's review of the first volume in the series, *Fundamental Algorithms*. Next up is a look at the final title, *Sorting and Searching*.

References
The Art of Computer Programming by Donald E Knuth; (Addison-Wesley Publishing Company).
Volume 1. Fundamental Algorithms.
Volume 2. Seminumerical Algorithms.
Volume 3. Sorting and Searching.

mar 85 6(3) 5 of 5.

It takes all sorts...

If you think that every important aspect of programming arises somewhere in the context of sorting and searching, then you're in full agreement with master programmer Donald Knuth. Mike Liardet looks at Sorting and Searching, the third volume of his book, "The Art of Computing Programming."

Sorting is the process of arranging things in ascending or descending order. Knuth points out that the layman's use of 'sorting' means something slightly different, and computer sorting could more correctly be called 'ordering' or 'sequencing'. In some senses sorting is related to the other topic in the volume, searching. This is because searching becomes a great deal easier once the items are sorted; imagine looking for a word in an unsorted dictionary.

Knuth divides sorting into two categories, internal and external. Internal sorting is used when all the data can be accommodated in high-speed internal memory, or RAM. External sorting is used when some, or most, of the data lies in external memory such as disks or tape, which are a great deal slower to access. This difference in access speed necessitates different approaches to the two types of sorting.

A prerequisite for external sorting is an ability to do internal sorting. However, as external sorting strategies are influenced by the hardware available and are generally more complex, I'll stick to internal sorting.

For internal sorting, Knuth presents well over two dozen different algorithms. It's not easy to pick a 'best' one, since different algorithms are better in different situations. It is, however, fairly easy to identify the worst one, called 'bubble sort'. For some perverse reason bubble sorting has enormous popularity with programmers, possible because it's easy to remember when Knuth's volume is not to hand. I'll present some of the more highly recommended routines.

A program that will enable you to test four of Knuth's sorting algorithms in a variety of different circumstances is listed in Fig 1. In all cases the data, or rather 'keys' to be sorted, are integer values in the array K (). For each key in K (), there is an associated record in the array R$(). For most sorting applications, it is not merely sufficient to sort the keys, but also the associated records: a telephone directory with the names sorted but the numbers in the original order would be quite useless! In some actual applications the keys may be an integral part of the record or they may be textual, and so on. Once you understand the algorithms it is relatively easy to tailor them to fit the specific sorting problem.

```
10000 REM INITIALIZE
10010 INPUT"NUMBER OF RECORDS TO SORT";N
10015 IF N<0 OR INT(N)<>N THEN PRINT CHR$(7):GOTO 10010
10020 INPUT"RANGE OF KEYS 1 TO..(OR 0 = IN ORDER, -1 = REVERSE ORDER)";HI
10025 IF HI<-1 OR HI>1E+08 OR INT(HI)<>HI THEN PRINT CHR$(7):GOTO 10020
10030 GOSUB 30000
10040 PRINT"SORT METHOD..."
10045 PRINT"  0. STOP"
10050 PRINT"  1. INSERTION SORT"
10060 PRINT"  2. SHELL'S SORT"
10070 PRINT"  3. QUICKSORT"
10080 PRINT"  4. DISTRIBUTION COUNTING"
10100 INPUT"ENTER 0 TO 4";CH
10105 IF CH<0 OR CH>4 OR INT(CH)<>CH THEN PRINT CHR$(7):GOTO 10100
10107 IF CH=0 THEN STOP
10110 ON CH GOSUB 20000,21000,22000,23000
10115 PRINT CHR$(7);CHR$(7);CHR$(7);
10120 GOSUB 31000
10130 RUN
20000 REM INSERTION SORTING - N ITEMS IN K() AND R$()
20010 FOR J=2 TO N
20020 K=K(J):R$=R$(J)
20030 FOR I=J-1 TO 1 STEP -1
20040 IF K>=K(I) THEN 20080
20050 K(I+1)=K(I):R$(I+1)=R$(I)
20060 NEXT I
20070 I=0
20080 K(I+1)=K:R$(I+1)=R$
20090 NEXT J
20100 RETURN
21000 REM SHELL'S SORT
21010 DIM H(15)
21020 H(1)=1
21030 FOR I=2 TO 15
21040 H(I)=3*H(I-1)+1:IF H(I)>=N THEN 21070
21050 NEXT I
21060 PRINT"N>7174453 ERROR":STOP
21070 T=I-2:IF T<1 THEN T=1
21080 FOR S=T TO 1 STEP -1
21090 H=H(S)
21100 FOR J=H+1 TO N
21110 K=K(J):R$=R$(J)
21120 FOR I=J-H TO 1 STEP -H
21130 IF K>=K(I) THEN 21170
21140 K(I+H)=K(I):R$(I+H)=R$(I)
21150 NEXT I
21160 REM ASSUMES I IS DEFINED ON COMPLETION OF FOR-LOOP
21170 K(I+H)=K:R$(I+H)=R$
21180 NEXT J
21190 NEXT S
```


Apr 85 6(4) 1 of 5.

```
21200 RETURN
22000 REM QUICKSORT WITH INSERTION SORTS FOR M OR LESS ITEMS
22010 DIM LSTACK(20),RSTACK(20)
22015 INPUT "M VALUE (EG 9):";M
22020 IF N<=M THEN 22150
22030 TOP=0:LFT=1:RGHT=N:K(0)=-1E+10:K(N+1)=1E+10:REM !!!GOTO 22040
22031 LFT=0:RGHT=N+1
22032 SWAP K(INT((LFT+RGHT)/2)),K(LFT+1):SWAP R$(INT((LFT+RGHT)/2)),R$(LFT+1)
22034 IF K(LFT)>K(LFT+1)THEN SWAP K(LFT),K(LFT+1):SWAP R$(LFT),R$(LFT+1)
22036 IF K(LFT+1)>K(RGHT)THEN SWAP K(LFT+1),K(RGHT):SWAP R$(LFT+1),R$(RGHT)
22038 IF K(LFT)>K(LFT+1)THEN SWAP K(LFT),K(LFT+1):SWAP R$(LFT),R$(LFT+1)
22039 LFT=LFT+1:RGHT=RGHT-1
22040 K=K(LFT):R$=R$(LFT)


22050 I=LFT:J=RGHT+1
22060 I=I+1:IF K(I)<K THEN 22060
22070 J=J-1:IF K<K(J) THEN 22070
22080 IF J>I THEN SWAP K(I),K(J):SWAP R$(I),R$(J):GOTO 22060
22090 SWAP K(LFT),K(J):SWAP R$(LFT),R$(J)
22100 IF RGHT-J>=J-LFT AND J-LFT>M THEN TOP=TOP+1:LSTACK(TOP)=J+1:RSTACK(TOP)=RGHT:RGHT=J-1:GOTO 22040
22110 IF J-LFT>RGHT-J AND RGHT-J>M THEN TOP=TOP+1:LSTACK(TOP)=LFT:RSTACK(TOP)=J-1:LFT=J+1:GOTO 22040
22120 IF RGHT-J>M AND M>=J-LFT THEN LFT=J+1:GOTO 22040
22130 IF J-LFT>M AND M>=RGHT-J THEN RGHT=J-1:GOTO 22040
22140 IF TOP>0 THEN LFT=LSTACK(TOP):RGHT=RSTACK(TOP):TOP=TOP-1:GOTO 22040
22150 IF M>1 THEN GOSUB 20000
22160 RETURN
23000 REM DISTRIBUTION COUNTING
23010 IF HI>1000 THEN PRINT CHR$(7);'KEY RANGE > 1000!!":RUN
23020 U=1:V=HI
23030 DIM COUNT(V-U)
23040 FOR I=0 TO V-U:COUNT(I)=0:NEXT I
23050 FOR J=1 TO N:COUNT(K(J)-U)=COUNT(K(J)-U)+1:NEXT J
23060 FOR I=1 TO V-U:COUNT(I)=COUNT(I)+COUNT(I-1):NEXT I
23065 R=N

23070 IF R=0 THEN RETURN
23080 IF COUNT(K(R)-U)<R THEN R=R-1:GOTO 23070
23090 IF COUNT(K(R)-U)=R THEN COUNT(K(R)-U)=COUNT(K(R)-U)-1:R=R-1:GOTO 23070
23100 R$=R$(R):K=K(R):J=COUNT(K(R)-U):COUNT(K(R)-U)=COUNT(K(R)-U)-1
23110 S$=R$(J):S=K(J):L=COUNT(K(J)-U):COUNT(K(J)-U)=L-1:R$(J)=R$:K(J)=K:R$=S$:K=S:J=L:IF J<>R THEN 23110
23120 R$(J)=R$:K(J)=K:R=R-1:GOTO 23070
30000 REM SET UP K() AND R$() WITH N VALUES DETERMINED BY HI
30010 DIM K(N+1),R$(N)
30020 FOR I=1 TO N
30030 IF HI>0 THEN K(I)=INT(RND(1)*HI+1)
30040 IF HI=0 THEN K(I)=I
30050 IF HI=-1 THEN K(I)=N-I+1
30060 R$(I)=STR$(K(I))
30070 NEXT I
30080 IF HI<=0 THEN HI=N

30090 RETURN
31000 REM CHECK K() IS SORTED AND R$() IS IN STEP WITH IT
31010 FOR I=1 TO N
31020 PRINT I,K(I)
31030 IF K(I)<>VAL(R$(I)) THEN PRINT"^^RECORD ERROR^^"
31040 IF I=1 THEN 31060
31050 IF K(I)<K(I-1) THEN PRINT"^^ORDER ERROR^^"
31060 NEXT I
31070 RETURN
```

Fig 1 Sorting program

Line 10000 — Initialisation and menu control
Line 20000 — Insertion sorting routine
Line 21000 — Shell's sorting routine
Line 22000 — Quicksort routine
Line 23000 — Distribution counting routine
Line 30000 — Routine to initialise data to be sorted
Line 31000 — Routine to print and check sorted data

Fig 2 Sorting program structure

The program is structured as shown in Fig 2.

The simplest sorting algorithm is called 'insertion sorting'. Imagine a situation where the list of keys is partitioned in two, with a sequence of keys in order up to a given point, and thereafter out of order. For example:

2 3 5 6 4 8 9 7 1

By scanning the values to the left of the marked key, we can gradually move these values one place right until we arrive at the right place to insert the marked key. This increases the size of the sorted partition by one. The above example would become:

2 3 4 5 6 8 9 7 1

By repeatedly applying this method, the sorted partition grows until all the keys are sorted. To get it started, only the first key is deemed to be sorted since any single value must be 'sorted', no matter what it is. Initially all keys, bar the first, are in the unsorted partition. In the Basic routine (Fig 1, line 20000) the variable J marks the boundary between the two partitions, and K is used to hold the key to be inserted — it cannot be left *in situ*, as it would be overwritten by the shuffling up to accommodate it.

Shell sorting was devised by Donald L Shell in 1959. In some sorting algorithms, the keys are only moved short distances at a time; this can be highly inefficient if the keys have to move a long way. Shell's method 'encourages' the keys to move in long jumps initially, and it then works out the details later by successively shorter jumps, or 'increments'. If the increments are successively 4, 2 and 1, the following nine keys would be sorted as follows:

```
        5 7 9 4 3 1 2 6 8
        *       *       *
4-sort .
        3 1 2 4 5 7 9 6 8
        *   *   *   *   *
2-sort .
        2 1 3 4 5 6 8 7 9
        * * * * * * * * *
1-sort .
        1 2 3 4 5 6 7 8 9
```

In effect the 4-sort does an insertion sort on four independent sequences of keys, where in each sequence the keys are four apart. The first of these sequences (marked with asterisks) comprises the keys 5, 3 and 8. The second comprises the keys 7 and 1, and so on. Note that all four of these sequences are correctly sorted following the 4-sort. The 2-sort does the same thing for just two sequences, with keys two apart. Finally, the 1-sort sorts a single sequence of adjacent keys and gets everything in the right order. In fact, the 1-sort is identical to the insertion sort.

Any sequence ending with 1 will work. (Insertion sorting is a special case of the method with a single increment of 1 being used.) In fact powers of 2 provide a fairly

Apr 85 6(4) 2 of 5.

		500 keys			20 keys
Range of keys	1. .500	1. .10	1. .500	1. .500	1. .20
Init ordering	Random	Random	~~In order~~	Reversed	Random
Insertion	16: 17	14:37	0:14	>20:00	1.5
Shell	1:55	1:26	1:03	1.35	1.5
Quicksort (M=9)	1:01	0:50	$$$$	$$$$	1.5
Quicksort (M=1)	1:03	0:58	$$$$	$$$$	1.5
Quicksort + (M=9)	1:00	0:51	7:30	7:04	1.5
Quicksort + (M=1)	1:02	0:59	7:18	6:52	1.5
Distr. Counting	0:34	0:27	0:26	0:34	1.5

Fig 3 Performance of the sort routines

```
35000 REM BINARY SEARCH FOR K IN N KEYS IN K(), RETURNS POSN (=-1 FOR FAILURE)
35010 L=1:U=N
35020 IF U<L THEN POSN =-1:RETURN
35030 POSN=INT((L+U)/2)
35040 IF K<K(POSN)THEN U=POSN-1:GOTO 35020
35050 IF K>K(POSN)THEN L=POSN+1:GOTO 35020
35060 RETURN
```

Fig 4 Binary search routine

poor performance, and after extensive analysis Knuth suggests some better alternatives. One of these is the sequence used in the routine here (Fig 1, line 21000). The increments are produced from the expression (3^K — 1)/2, with values of K decreasing from some initial value down to 0. (The code given does calculate these values but without recourse to exponentiation, and the increments are held in the array H().) The initial value used is the largest possible, not exceeding one third of the number of items to be sorted. For example, to sort 1000 keys the increments would be 121, 40, 13, 4 and 1.

Quicksort

The Quicksort method was devised by C A R Hoare in 1962. This is one of the more complex methods to code (Fig 1, line 22000) particularly if the implementation language is not recursive — as is the case with Basic. In its basic form, a list of keys is sorted by choosing the first key as a 'pivot' and then dividing the remaining keys into two partitions: keys to the left being less than, or equal to, the pivot and to the right being greater than, or equal to, the pivot. To obtain these two partitions we scan right from the first key after the pivot until we find a 'rogue' key (greater than the pivot) and scan left from the end until we find another rogue key (less than the pivot). These keys can then be swapped, and this continues until the right scan crosses the left: this is the correct position for the pivot element. For example, quicksorting the following numbers:

5 7 9 4 3 1 2 6 8

Exchange 7 and 2.

5 2 9 4 3 1 7 6 8

Exchange 9 and 1.

5 2 1 4 3 9 7 6 8

Place pivot (exchange 5 and 3) . .

3 2 1 4 5 9 7 6 8

At this point the 5 is correctly placed; all the values to the left of it are less than it, and all those to the right are greater. Sorting these two partitions can be seen as two separate independent problems, so we can continue by quicksorting 3, 2, 1 and 4, and then quicksorting 9, 7, 6 and 8, and so on.

There are various refinements to this method. As insertion sorting is generally regarded as the most efficient method for small lists, we can invoke insertion sorting instead of quicksorting when the lists get below a particular size (the value M in Fig 1 at line 22000). There's nothing to lose by abandoning the sorting when a list gets below size M, and then calling insertion sorting just once for the whole list, right at the end. Note that if M is 1, then pure quicksorting is used.

A major problem with quicksorting is that it's at its worst when the list is already sorted. Unlike most methods, it's at its best when the keys are scrambled. This seems very unsatisfactory, and can be corrected to some degree by arranging for a more careful choice of pivot. The method recommended by Knuth is to first interchange the second and middle keys in the list, then sort just the first, second and last keys, pivoting on the middle one. For the aforementioned sequence:

5 7 9 4 3 1 2 6 8

Swap the second and middle.

5 3 9 4 7 1 2 6 8

Sort first, second and last only. .

3 5 9 4 7 1 2 6 8

Now partition the third to last keys using 5 as the pivot .

3 5 2 4 1 7 9 6 8

Insert pivot in the right position. .

3 1 2 4 5 7 9 6 8

This procedure makes little difference to randomly ordered keys, and considerably improves the situation if the keys are already ordered.

Both these enhancements are incorporated in the routine at line 22000. The routine prompts for a suitable value of M before starting: Knuth recommends 9 as optimum, although the best value depends on the characteristics of the programming language you are using. Lines 22031 to 22039 make a careful selection of the pivot. Simple pivot selection is obtained by deleting the REM at line 22030.

In circumstances where the keys are numeric and have a restricted range of values, a very efficient sorting procedure can be applied by noting the frequency of occurrence of each key. This is the strategy adopted by 'distribution counting' sorting. The first phase of the algorithm obtains the number of occurrences for each key. In Fig 1 at 23000, if the lowest key value is U and the highest is V, then COUNT (0) holds the number of occurrences of U, and COUNT (V-U) holds the number of occurrences of V. For example, the counts for the 2 3 1 1 3 2 1 2 2 would be:

1-count: 3
2-count: 4
3-count: 2

Once sorted, we will see 3 '1's followed by 4 '2's, followed by 2 '3's. If each of the counts is now accumulated, for example, the 2-count becomes 3+4 and the 3-count becomes 3+4+2, then the value in each count will indicate the last position for each of the corresponding keys:

1-count: 3
2-count: 7
3-count: 9

So the ' 1s will appear in position 1 to 3, the '2's in 4 to 7, and the '3's in 8 to 9.

Now, scanning the numbers from right to left, we search for a key which is too far to the left:

*
2 3 1 1 3 2 1 2 2

The totals in the counts make this test relatively easy, and the found key can be inserted at the position indicated by its count (position 9):

2 3 1 1 2 1 2 2 3

By adjusting the counts and repeating this process, it is then possible to get all the keys into the correct order. Fig 1 (at line 23000) contains extra sophistications which further minimise the amount of scanning and moving needed to sort the keys.

Apr 85 6(4) 3 of 5

In order to assess how effective these different algorithms are, Fig 3 outlines the results of running each of them under various conditions. The times are in minutes and seconds (obtained in interpreted Microsoft Basic on an Apricot — some appreciation of the performances can be gained by noting that it takes all of 12 seconds just to initialise the data for 500 keys). $ signs indicate times definitely in excess of 10 minutes and estimated to be about one hour, demonstrating the appalling behaviour of standard Quicksort if the keys are ordered. The following conclusions can be drawn.

Insertion sorting is good for short-lists but hopeless for long ones, unless the list is already, or very nearly, in order. (All methods appear equal for short lists in Fig 3, but this is due to inadequacies in my reflexes.) This is the only method considered here that maintains equal keys in their original order — this can be important for some applications.

Shell sorting performed well on all tests, with consistent response times no matter what the state of the input.

Quicksorting is excellent for random lists, but no use for ordered lists. Pure quicksorting (when M=1) is slightly slower than quicksort combined with insertion sorting. More careful selection of a pivot value mitigates the ordered list problem.

Distribution counting was best all round, but is not universally applicable.

If asked to nominate a good, general-purpose, workhorse sort routine I would choose Shell sorting. In fact it would not be difficult to write a super-sort procedure which, from a preliminary scan of the data, could choose the most appropriate routine. Knuth covers another 20 or so possible algorithms.

Searching

Of the two topics, Knuth gives far more prominence and material to sorting. Searching is concerned with retrieving data that has been stored with a given identification. The identification is the 'key', and the data is the associated 'record'.

Sequential searching is the most obvious technique for searching a list: start at the front, and keep going until either you find the key you want or reach the end. On average half the keys are scanned for a successful search, and all of them are scanned for an unsuccessful one.

A more efficient technique, which is almost as simple to implement, is called binary search (Fig 4); this only works if the list is in order. Given an ordered list of keys, examine the middle one, which will either be greater, less, or equal to the key we are seeking.

If it's equal then we have successfully found the key. If it's less than the given key, then we can continue searching for the key in the right half of the list, otherwise continue on the left. The search terminates unsuccessfully when there is no list left, when lower pointer L exceeds pointer U in the routine given.

As it's a more efficient technique, binary search can be blindingly fast even for very long lists. A maximum of 20 comparisons would be made to search a million keys — quite an improvement on straight sequential search. Marginal improvements have been suggested — not examining the middle element every time, but making a more careful choice determined by the key we are seeking. In practice, the increase in complexity offsets any other gains.

Binary trees

Frequently, following an unsuccessful search, we may wish to insert the unfound key. If we are using binary search, then this can be computationally expensive for long lists of keys. If, instead of storing the keys sequentially, a 'binary tree' structure can be used, then binary search and easy insertion can coexist (Fig 5). The price for this is that the tree requires slightly more storage and is more complex to scan.

A binary tree is built up of 'nodes'. Each node contains the text of one key, and

```
50000 REM BUILDS AND SEARCHES A BINARY TREE
50010 DIM KEY$(1000),BEFORE(1000),AFTER(1000)
50020 KEY$(1)="ROOT":BEFORE(1)=0:AFTER(1)=0
50030 AVAIL=2
50040 INPUT"TYPE A KEY";KEY$
50050 GOSUB 51000:IF FOUND=1 THEN PRINT"ITS AT NODE ";NODE:GOTO 50040
50060 PRINT"NOT FOUND - INSERTING IT"
50070 GOSUB 52000
50080 PRINT"AND ITS AT NODE ";NNODE:GOTO 50040
51000 REM SEARCH BINARY TREE SETS FOUND AND NODE
51010 NODE=1:REM START AT ROOT
51020 IF KEY$=KEY$(NODE)THEN FOUND=1:RETURN
51030 IF KEY$<KEY$(NODE) AND BEFORE(NODE)<>0 THEN NODE=BEFORE(NODE):GOTO 51020
51040 IF KEY$>KEY$(NODE) AND AFTER(NODE)<>0 THEN NODE=AFTER(NODE):GOTO 51020
51050 FOUND=0:REM FAILURE (BUT NODE SET FOR INSERTION)
51060 RETURN
52000 REM (FOLLOWING UNSUCCESSFUL SEARCH) CREATE AND INSERT A NODE FOR KEY$ BEFORE/AFTER NODE
52010 IF AVAIL>1000 THEN PRINT"STORAGE OVERFLOW!":STOP
52020 KEY$(AVAIL)=KEY$:BEFORE(AVAIL)=0:AFTER(AVAIL)=0:NNODE=AVAIL:AVAIL=AVAIL+1
52030 IF KEY$<KEY$(NODE)THEN BEFORE(NODE)=NNODE:RETURN
52040 AFTER(NODE)=NNODE:RETURN
```

Fig 5 Binary tree program

```
40000 REM SOUNDEX FOR ANY X$ <>"" AND CONTAINING ONLY "A" TO "Z"
40010 DATA "AEHIOUWY","BFPV","CGJKQSXZ","DT","L","MN","R"
40020 DIM GROUP$(6):FOR I=0 TO 6:READ GROUP$(I):NEXT I
40030 REM GROUP$(0) IGNORED, OTHERWISE A LETTER IN GROUP$(I) HAS DIGIT I
40040 SNDEX$=LEFT$(X$,1):REM FIRST LETTERS OF SOUNDEX AND X$ ARE THE SAME
40050 C$=SNDEX$:GOSUB 41000:OVAL$=VALU$
40060 FOR I=2 TO LEN(X$)
40070 C$=MID$(X$,I,1):GOSUB 41000
40080 IF VALU$="0" OR VALU$=OVAL$ THEN 40100:REM SKIP "VOWELS" AND "REPETITIONS"
40090 SNDEX$=SNDEX$+VALU$
40100 OVAL$=VALU$
40110 NEXT I
40120 SNDEX$=LEFT$(SNDEX$+"000",4):REM TRUNCATE/PAD WITH TRAILING 0S
40130 RETURN
41000 REM RETURN VAL$ FOR GIVEN C$ + CHECK FOR ILLEGAL LETTERS
41010 FOR VALU=0 TO 6
41020 CHS$=GROUP$(VALU)
41030 FOR J=1 TO LEN(CHS$)
41040 IF C$=MID$(CHS$,J,1)THEN VALU$=CHR$(ASC("0")+VALU):RETURN
41050 NEXT J
41060 NEXT VALU
41070 PRINT"ILLEGAL LETTER: ";C$:STOP
```

Fig 6 Soundex routine

Apr 85 6(4) 4 of 5.

pointers to the before and after nodes. (In real applications there may be other information as well.) These pointers reference other nodes from which all the words before or after the current node can be accessed, if there are no other nodes, the pointers are simply 'null'.

A binary tree is searched, starting at the root node. If this node contains the key then we have found the place we want. Otherwise the key must be before or after the current node, and we move to the next node accordingly and repeat the process.

If there is no next node then the key is not in the tree, and we can insert it at this point if necessary.

This method works best with storage management routines to allocate and deallocate storage as nodes (that is, keys) are added and deleted. In the routine given here only minimal storage management is attempted to keep things simple.

In some cases a binary tree can become unbalanced. The worst case occurs if the keys are inserted in order, when the algorithm just performs an unnecessarily complex sequential search. If the keys are presented in a suitably random order, then all the branches will be at roughly the same depth. Knuth also presents techniques for keeping trees well balanced.

Throughout this analysis we have assumed that it is readily possible to identify two keys as being equal. But when working on an interactive system, it can sometimes be a problem to recall the precise spelling of a word, such as a surname. Knuth presents a technique, called soundexing, which can convert similar sounding words into the same key (Fig 6). The technique was developed by Margaret Odell in 1918, predating computers by a good many years. Essentially the method converts any word into a key, consisting of a letter followed by three digits. Similar sounding letters are assigned the same digit; vowels and a few other letters are ignored altogether, as are repeated letters.

Conclusion

This concludes my presentation of Knuth's three volumes on *The Art of Computer Programming*. It should be remembered that these volumes run to over two thousand pages in total, so I have had to be highly selective as to which material I have featured.

Unfortunately many interesting and pertinent algorithms have fallen by the wayside, and if my writings have whetted your appetite for more information then you will have to buy the volumes to find out more.

References
The Art of Computer Programming, by Donald E Knuth; Addison-Wesley Publishing Company.
Volume 1 Fundamental Algorithms
Volume 2 Seminumerical Algorithms
Volume 3 Sorting and Searching

Apr 85 6(4) 5 of 5.

The art of programming slowly

Recently I bought the first three volumes in the series *The Art of Computer Programming*. What has happened to the other four volumes in the series? Although the preface to the first volume suggests that the author finished writing all seven volumes in 1967, the third was only published in 1973. Do you know if and when the remaining volumes will be published?
G Hjaltson.

The publisher Addison Wesley says that work on the series is still taking place, and it hopes to release volume four some time in 1986.

Donald Knuth, the author, seems to have become the victim of his own success with this series. The first brilliant and exceptionally comprehensive volumes have set a very high standard for the rest.

In 1967 the computer industry was barely 20 years old, and a comprehensive programming guide may have seemed feasible. Since then, computing has advanced at such a rate that it is difficult to keep a monthly magazine up-to-date, let alone a series of books. It's hard to see how Knuth can encapsulate the rest of computing knowledge in four volumes.
SG

APC Oct 85 6(10) p 82.

Multiple comments

You are to be congratulated on your recent series of reviews by Mike Liardet of Knuth's "The Art of Computer Programming". These types of books and reviews can only lead to a more efficient approach to programming, and a greater utilisation of equipment. It is also pleasing to see some more general notes about programming, rather than just machine specific programs, which are largely undocumented. In that spirit, I wish to make some comments on the binary search routine contained in figure 4 on page 82 of the April, 1985 issue. That routine will not necessarily find the first or leftmost match if the list contains more than one possible match. If, for example, the list contains the following data:

1 2 3 3 3 4 4 5 6

and the search is for "3", it will find a "3", but not necessarily the first "3". It is necessary to cover the possibility of potential multiple matches. Because the list is sorted, any possible multiple matches will be adjoining, so a fan should be performed to find multiple matches. One way of doing this is by inserting the following code into figure 4 of that article. See listing 1.

```
35052 LPOSN = POSN : UPOSN = POSN : REM Store found location
35053 LPOSN = LPOSN - 1 : IF LPOSN < 1 THEN 35055 : REM go down
35054 IF K = K(LPOSN) THEN 35053 : REM Further match. Try again
35055 UPOSN = UPOSN + 1 : IF UPOSN > N THEN 35058
35056 REM Execution drops to here if no further lower, and starts to go up
35057 IF K = K(UPOSN) THEN 35055 : REM Further match. Try again
35058 REM This returns the following values in K
35059 REM POSN = -1 : no matches   /  (LPOSN+1 = POSN) AND (UPOSN-1 = POSN)
      : 1 match at POSN  /  ELSE multiple matches in the range LPOSN+1 TO UPOSN-1
```

Listing 1

Of course, the calling routine would have to be adjusted to take these alterations into account.

It should also be noted that the binary search routine will also work on a list that is not sorted, but indexed. This may be useful in some situations, for example if it is not desired to move strings around in memory (to avoid the garbage collection routines) or if more than one array is used for related data.

A sample program to do this is in listing 2.

J Lamich

```
10 CLS : DIM K$(10),K(10) : REM BINARY SEARCH DEMONSTRATION
20 N = 10
30 FOR I = 1 TO N : READ K$(I) : NEXT
50 DATA E,D,J,T,L,J,B,Z,A,L
60 PRINT "Unsorted list "
70 FOR I = 1 TO N
80 PRINT K$(I);
90 NEXT
100 PRINT
110 REM ======= sort
120 FOR A = 1 TO N
130 P = 1
140 FOR B = 1 TO N
150 IF K$(A) > K$(B) THEN P=P+1
160 IF K$(A) = K$(B) AND A>B THEN P=P+1
170 NEXT
180 K(P) = A
190 NEXT
200 PRINT
210 PRINT "Sorted list"
220 FOR I = 1 TO N
230 PRINT K$(K(I));
240 NEXT
250 PRINT
260 INPUT "Target ";K$
270 GOSUB 35000
280 IF POSN = -1 THEN PRINT "Not found ": GOTO 200
290 IF (LPOSN+1 = POSN) AND (UPOSN-1 = POSN) THEN PRINT "One match at " POSN : GOTO
```

```
200
300 PRINT "Multiple Matches at "LPOSN+1" to "UPOSN-1
310 GOTO 200
35000 REM binary search
35010 L = 1 : U = N
35020 IF U < L THEN POSN = -1 : RETURN
35030 POSN = INT((U+L)/2)
35040 IF K$ < K$(K(POSN)) THEN U = POSN -1 : GOTO 35020
35050 IF K$ > K$(K(POSN)) THEN L = POSN +1 : GOTO 35020
35052 LPOSN = POSN : UPOSN = POSN : REM Store found location
35053 LPOSN = LPOSN - 1 : IF LPOSN < 1 THEN 35055 : REM go down
35054 IF K$ = K$(K(LPOSN)) THEN 35053 : REM Further match. Try again
35055 UPOSN = UPOSN + 1 : IF UPOSN > N THEN 35060
35056 REM Execution drops to here if no further lower, and starts to go up
35057 IF K$ = K$(K(UPOSN)) THEN 35055 : REM Further match. Try again
35060 RETURN
```

Listing 2

APC Jun 85 6(6)
p 170-171.

Soughts of sorts

I read with great interest your recent Mike Liardet series review of Donald Knuth's "The Art of Computer Programming". However, in view of its rather spectacular performance against other sorts and because I had never even heard of the distribution sort before, I was more than a little disappointed that Mr Liardet dismissed it with the comment that it was "not universally applicable".

One of the monotonously repetitive tasks of my computer (and, I'm sure, of many of your readers') is the sorting of string arrays in which the keys are the string elements themselves. I'm therefore constantly on the lookout for a faster sort routine and immediately determined to try to harness the distribution sort.

After a full day of hacking (and tearing my hair out!) I finally settled on the following algorithm as being probably optimal:

1 the use of the distribution sort as a presort of a randomly-ordered string array to arrange the array so that all string elements would be clustered with others sharing the same initial (naturally in ascending order); and then

2 the use of an "intelligent" bubble sort to arrange each cluster of elements in lexicographic order.

The following program, written on a Commodore 64, generates a randomly-ordered array of strings, each of 30 characters' length, and applies this distribution/bubble sort technique to it. The TI$ reference is, of course, the onboard C64 timer; interested readers using other computers will have to work out their own timing devices.

I have chosen 300 as the size of the array because, as the following Benchmark chart shows, that size appears to be the break-even point between my sort and the old faithful Quicksort. However, my sort has the advantage that, if the randomly-ordered array just happens to be more or less truly ordered to begin with, it will perform considerably faster, whereas — as everyone knows — the Quicksort will be disastrously slower in such a circumstance.

For instance, one Quicksort run I performed on a 500-element array took 9 minutes 45 seconds as opposed to the mean 4 minutes 50 seconds shown in the chart shown, leading me to suspect that that particular randomly-ordered array was not as random as it might have been!

I do hope these observations will be of interest to some of your readers.

K Riordan

APC Jun 85 6(6) p 171 and 173.

```
1  N=300 : DIM A$(N),C(26) : FOR X=1 TO N : FOR
   Y=1 TO 30
2  Z=INT(RND(0)*26)+65 : A$(X)=A$(X)+CHR$(Z) :
   NEXT : NEXT
3  TI$="000000" : PRINT TI$" SORTING . . . "
4  FOR J=1 TO N : A=ASC(A$(J))-64 : C(A)=C(A)+1
   : NEXT
5  FOR J=2 TO 26 : C(J)=C(J)+C(J-1) : NEXT :
   R=N+1
6  R=R-1 : IF R=0 THEN 13
7  A=ASC(A$(R))-64 : IF C(A)<R THEN 6
8  IF C(A)=R THEN C(A)=C(A)-1 : GOTO 6
9  A$=A$(R) : J=C(A) : C(A)=C(A)-1
10 B$=A$(J) : A=ASC(B$)-64 : K=C(A) : C(A)=C(A)-1
   : A$(J)=A$ A$=B$ : J=K : IF J<>R THEN 10
11 A$(J)=A$ : GOTO 6
12 REM *** NOW FOR THE "INTELLIGENT" BUBBLE
   SORT! ***
13 FOR J=1 TO N-1 : FOR K=J+1 TO N
14 IF ASC (A$(J))<ASC(A$(K)) THEN K=N : GOTO 16
15 IF A$(J)>A$(K) THEN A$=A$(J) : A$(J)=
   A$(K)=A$(K)=A$
16 NEXT : NEXT : PRINT TI$" SORTED!" : END
```

The Benchmarks:

# OF ELEMENTS	MY SORT	QUICKSORT
50	00′ 05″	00′ 08″
100	00′ 14″	00′ 20″
150	00′ 24″	00′ 41″
200	00′ 45″	00′ 55″
250	01′ 09″	01′ 17″
300	01′ 45″	01′ 45″
500	07′ 05″	04′ 50″

SORTING OUT THE SORTS

The program listed with this article was developed to test the speed and efficiency of four sorting algorithms: Insertion, Shell, Quick and Selection. The program will probably run as is in most BASICs.

What's usually required of a sort is to put a list of names into alphabetical order, but the average textbook seems to present sorts for numbers with no indication of the best choice for a particular task.

The choice of sorting algorithms is broad: more than three dozen are known, spread across some hundred texts. The most popularly presented, and the slowest if list size is more than 11, is the bubble sort.

Fortunately, the choice can be narrowed down to short algorithms which work in RAM only and are not bubbly sorts in disguise!

The program

The first program line, line 30, asks for the size of the array to be generated and sorted. Start with a choice of 10 to check the sorts are working as expected.

Lines 70-130 generate the required number of capital letters and place them in the array. The variable CD is set at 64, one less than the ASCII code for 'A'. A number is generated in line 90 and added to CD. If this number is acceptable, the character it represents is placed in array CH$. If not, C is decremented by one and the process is repeated. The loop runs until BN (big number!) letters are placed in the array. Line 130 prints out the unsorted list and could be omitted from the program.

The straight Insertion sort and the Shell (insertion algorithm) sort are particularly useful.

The Selection sort is always the slowest, since the same number of compares and swaps is made if the list is random, or if only one element is out of order. The Quick sort isn't much better if only a few elements are out of order. If the list is random with more than 500 elements Quick sort is useful, but only if there is no shortage of RAM to store the two extra arrays the sort requires.

The Shell sort is best for a random list, and the Insertion sort is best when only a few items in the list are out of order. For most work the Insertion sort will do. For instance, a mailing list is only truly random when first typed in, and thereafter only insertions need to be sorted. The Insertion sort is simplest to understand: search forward for an out-of-order element, then search back through the list and insert the element in its proper place. The Shell sort is a little more complicated: elements compared are a specified distance apart, which decreases until only adjacent elements are compared.

Mr Jankowski,
Timaru, New Zealand

```
10 REM SORT MEASURE by LZ Jankowski
15 REM COPYRIGHT July 1985
20 :
30 CLS: INPUT "# of items in list "; BN: CLS
40 DIM CH$(BN), A(BN), B(BN)
50 :
60 REM ---------Create random list of letters-----------------
70 CLS: PRINT TAB( 20)"* Programming *": CO=64
80 FOR C=1 TO BN: CO=CO+10: IF CO=94 THEN CO=64
90 :  X=INT(10*RND(1))+CO
100 :   IF X<65 OR X>90 THEN C=C-1: GOTO 120
110 :    CH$(C)=CHR$(X)
120 NEXT : CLS: PRINT "Random list is": PRINT : PRINT
130 FOR C=1 TO BN: PRINT C"=" CH$(C) SPC( 4);: NEXT : PRINT : PRINT
140 :
150 REM -------------------MENU-------------------------------
160 PRINT : PRINT : PRINT : PRINT TAB( 15)"1> Insertion Sort"
170 PRINT TAB( 15)"2> Shell     Sort"
180 PRINT TAB( 15)"3> Quick     Sort": PRINT TAB(15)"4> Selection Sort
190 PRINT : INPUT "Choice "; C
200 IF C<1 OR C>4 THEN RUN
210 :
220 PRINT !(28): PRINT "* Sorting *": PRINT : PRINT
230 ON C GOSUB 500, 400, 570, 750: GOSUB 350
240 :
250 REM ------Now sort 'A' from bottom to top of list---------
260 FOR C=1 TO 40: PRINT "-";: NEXT
270 PRINT : PRINT "Sorting 'A' from bottom to top of list": PRINT
280 GOSUB 500: GOSUB 350: GOSUB 400: GOSUB 350
290 GOSUB 570: GOSUB 350: GOSUB 750: GOSUB 350
300 :
310 PRINT : INPUT "RUN again ";Q$: IF LEFT$(Q$,1)="Y" THEN RUN
320 END
330 :
340 REM -------Print sorted list & # of compares & swaps------
350 FOR C=1 TO BN: PRINT C "=" CH$(C) SPC( 4);: NEXT
360 PRINT : PRINT : PRINT "Compares= "CM,"Swaps= "S: PRINT
370 CH$(BN)="A": CM=0: S=0: RETURN
380 :
390 REM ------Shell sort based on Insertion algorithm---------
400 I=(2^INT(LOG(BN)/LOG(2)))-1
410 I=INT(I/2)
420 IF I<1 THEN 470
430 FOR N=1 TO I: FOR C=N+I TO BN STEP I: M=C: C$=CH$(M)
440 CM=CM+1: IF CH$(M-I)<=C$ THEN 460
450 CH$(M)=CH$(M-I): S=S+1: M=M-I: IF M>I THEN 440
460 CH$(M)=C$: NEXT C: NEXT N: GOTO 410
470 PRINT "SHELL SORT": RETURN
480 :
490 REM --------------Insertion Sort--------------------------
500 FOR N=2 TO BN: M=N: C$=CH$(M)
510 CM=CM+1: IF CH$(M-1)<=C$ THEN 530
520 S=S+1: CH$(M)=CH$(M-1): M=M-1: IF M>1 THEN 510
530 CH$(M)=C$: NEXT
540 PRINT "INSERTION SORT": RETURN
550 :
560 REM ----Quicksort best for very long, random lists--------
570 SP=1: A(1)=1: B(1)=BN
580 FI=A(SP): SI=B(SP): SP=SP-1
590 SF=FI: SS=SI: C$=CH$(INT((FI+SI)/2))
600 CM=CM+1: IF CH$(SF)>=C$ THEN 630
610 SF=SF+1
620 GOTO 600
630 CM=CM+1: IF C$>=CH$(SS) THEN 650
640 SS=SS-1: GOTO 630
650 IF SF>SS THEN 670
660 S=S+1: E$=CH$(SF): CH$(SF)=CH$(SS): CH$(SS)=E$: SF=SF+1: SS=SS-1
670 IF SF<=SS THEN 600
680 IF SF>=SI THEN 700
690 SP=SP+1: A(SP)=SF: B(SP)=SI
700 SI=SS: IF FI<SI THEN 590
710 IF SP>0 THEN 580
720 PRINT "QUICKSORT": RETURN
730 :
740 REM ----Selection Sort, dreadful! CM=(N-1)*N/2 Swaps=N-1--
750 FOR N=1 TO BN-1:M=N: FOR C=N+1 TO BN:CM=CM+1:IFCH$(M)>CH$(C)THENM=C
760 NEXT: S=S+1: C$=CH$(N): CH$(N)=CH$(M): CH$(M)=C$: NEXT
770 PRINT "SELECTION SORT": RETURN
```

Your Computer Oct 85
p. 107-108.

Z80

The Z80 was designed by a group of one-time Intel employees who left and founded Zilog. The first product of this enterprise was and is one of the undisputed successes of its time. In addition to its definite technical advantages over the other micros of the time, it also gave the necessary impetus to one of the original 'standards' for small business computers by becoming the target CPU for CP/M. In truth CP/M was originally developed for the 8080A, but with the advent of the Z80 (a super-8080A, which does everything and more that an 8080A will do, including run 8080A code), subsequent CP/M developments tended to be directed more and more towards this powerful machine. It is interesting to consider in passing how many other of the processors which were introduced in 1977 are still being used in new machines in 1986. A brief look at the design of the ever-expanding Amstrad range soon shows that there is still plenty of mileage left in this device.

So far so good, but what is it that made the Z80 such a popular micro with system builders? One of the answers must undoubtedly be the amount of software support which is available for the Z80. In many ways this is an answer to the original criticism that the Z80 designers paid too high a price for retaining compatibility with the then very popular 8080A . With the benefit of 20-20 hindsight, the strong following which the Z80 gained from providing this 8080A compatibility, also gave it the vital headstart it needed in the second-generation 8-bit architecture race.

The Z80 runs from a single +5V supply, and uses only a single chip to provide all of the functions necessary for the CPU. The pin functions and assignments for the Z80-CPU are shown in Fig. 6.7.

The Z80 features minicomputer-style I/O and vectored interrupts. It has a large instruction set of 158 instructions, including the 78 instructions of the 8080A as a subset. These instructions provide extensive facilities for string, bit, byte and word operations. Block searches and block transfers, together with indexed and relative addressing result in very powerful data handling capabilities.

Duplicate sets of both general-purpose and flag registers are provided, easing the design and operation of control software thorugh rapid context switching. The programming model of the Z80 is shown in Fig. 6.8. There are essentially three groups of registers in the Z80. The first consist of duplicate sets of 8-bit registers; a principal set and an alternative set (indicated by the ' suffix). Both sets of registers have an accumulator, a flags register and six general-purpose registers. Transfer of data between these duplicate sets of registers is accomplished by means of "Exchange" instructions. The result is faster response to interrupts and easy, efficient implementation of such versatile techniques as background/foreground processing. The second set of registers have assigned functions: Interrupt Register (I), Refresh Register (R), Index Registers (IX and IY), Stack Pointer (SP), and the Program Counter (PC). The third group consists of two interrupt status flip-flops and two flip-flops to identify the current interrupt status mode. It is perhaps worth a brief look at some of the registers which may not be familiar from looking at other micros.

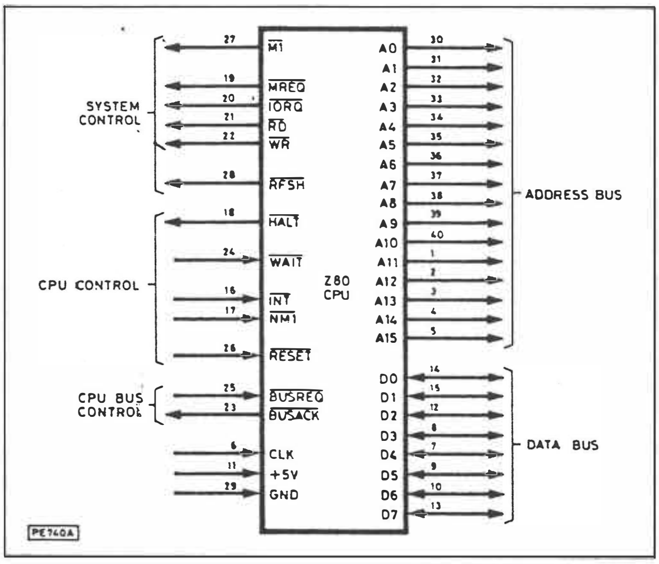


Fig. 6.7. Z80 pin functions and assignments

Memory Refresh: This register provides a user-transparent dynamic memory refresh capability. The lower 7 bits are automatically incremented, and all 8 are placed on the address bus during each instruction fetch cycle refresh time (i.e. when the $\overline{RFSH}$ signal output is low). This can be used as a refresh address to the system's dynamic memories, thereby simplifying system design.

Interrupt Register: This register holds the upper 8 bits of the memory address to be used in forming the 16-bit address to point to the table of addresses for the interrupt service routines. This register is used in servicing interrupts in mode 2, where the lower 8 bits of the address are provided by the interrupting peripheral device.

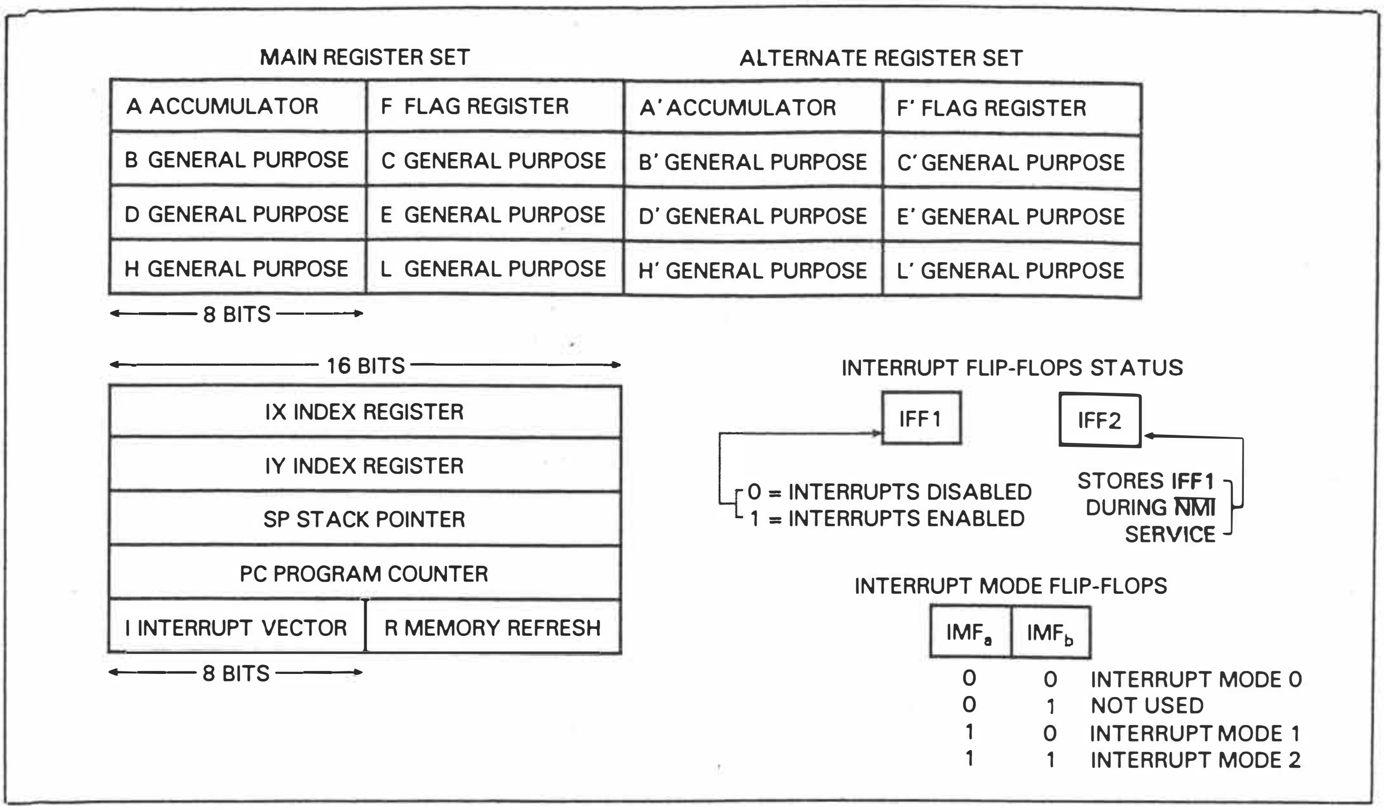


Fig. 6.8. Z80 programming model

Interrupt Mode: These flip-flops reflect the current interrupt mode, which may be 0, 1, or 2. Mode 0 is the 8080 mode, whereby the interrupting peripheral places an instruction on the bus. This is normally a restart instruction which will initiate a call to the selected one of eight restart locations in page zero of memory. Mode 1 is very similiar to the NMI mode, but it jumps to the code contained at location 0038 for its service routine (whereas an NMI uses location 0066). This mode is intended for non-Z80/8080 systems. Mode 2 is the flexible vectored mode described above, particularly intended to use the Z80 family and compatible peripheral devices most effectively.

Z80 PERIPHERALS

There are five major support peripherals which were designed specifically for the Z80. Instead of numbering these separately, it is common practise with the Z80 family to describe each device in terms of the family name, followed by the functional acronym (CPU, PIO, etc). Each device does, in fact, also have a conventional (different) part number, e.g. the standard Zilog Z80 CPU is the Z8400, or the Z8300 if from the low power family. The popular peripheral chips in the Z80 family are described briefly below.

Z80-PIO: The PIO (Parallel Input/Output) operates in both byte I/O transfer mode (with handshaking), and in bit mode (without handshaking). The PIO may be configured to interface with standard peripheral devices such as printers and keyboards. Typical part number: Z8420.

Z80-CTC: The CTC (Counter/Timer Circuit) features four programmable 8-bit counter/timers, each of which has an 8-bit prescaler. Each of the four channels may be configured to operate in either counter or timer mode. Typical part number: Z8430.

Z80-DMA: The DMA (Direct Memory Access) controller provides dual-port data transfer operations, and also has the ability to terminate data transfer as a result of pattern match in the transferred data. Typical part number: Z8410.

Z80-SIO: The SIO (Serial Input/Output) controller provides two channels. It is capable of operating in a variety of modes for both synchronous and asynchronus communications. Typical part number: Z8440.

Z80-DART: The DART (Dual Asynchronous Receiver/Transmitter) provides low cost asynchronous serial communication. It has two channels and a full modem control interface. Typical part number: Z8470.

CONCLUSION

This brings us to the end of our short series on the basics of micro systems. We hope that it has given enough of an insight into the workings of these fascinating machines to allow some sense to be made of the huge volumes of application data now available on the subject. As mentioned originally, a series such as this can hope to do little more than provide a general introduction to the subject. From here on the best course will depend very much how you wish to make use of of the basic technology.

The cost of providing a particular level of capability, counter to the natural law in most other spheres of endeavour, is likely to continue to fall for quite some time to come. The applications for micro system technology are generally limited only by the ingenuity of you, the designers, whilst the capabilities of the basic components are constantly being improved. The future for this technology therefore seems assured.

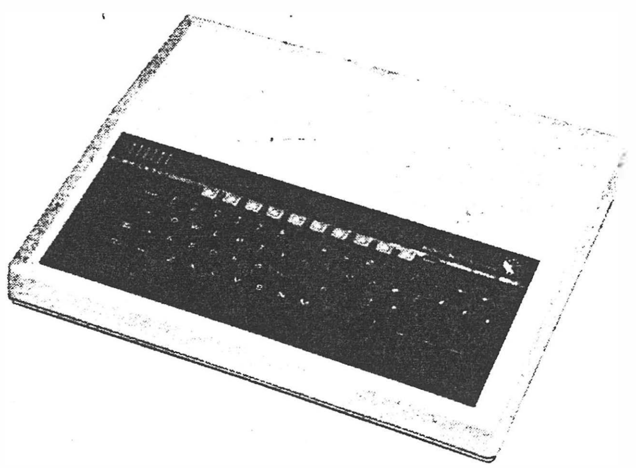


Photo illustrating the BBC Micro which employs a 6502 CPU as the main processor, but may use a Z80 as a second processor

TO COME: Next month in *PE* we will be outlining details of some constructional projects which will employ both the 6502 and Z80 microprocessors.

AEM	Australian Electronics Monthly	ETI	Electronics Today International
AHC	Australian Home Computers		
APC	Australian Personal Computer	M80	Micro-80
APH	Australian Photography		
AR	Amateur Radio		
ARA	Amateur Radio Action		
BB	Bits and Bytes (NZ)		
BI	Break In (NZ)		
BYC	Bumper Book of Programs by YC	MC	Micro Choice (UK)
CBA	CB Action		
CC	Creative Computing (US)	PCG	Personal Computer Games
CFG	Computer Fun and Games	PCN	Personal Computer News (UK)
CI	Computer Input (NZ)	PE	Practical Electronics (UK)
CLC	Classroom Computing	SYN	Sync (US)
CT	Computing Today (UK)	WM	Which Micro (UK)
CHC	Choice	YC	Your Computer
EA	Electronics Australia	YCU	Your Computer (UK)

FURTHER LITERATURE RELATING TO THE VZ200/300 COMPUTER

As an extension to my list of magazine articles, I have produced the following list of books (I have copies of all of the publications). The books relate to the VZ computer specifically, Microsoft BASIC Level II or the Z-80 microprocessors, as used in the VZ200/300. Additionally, I hold a lot of additional technical information, ROM listings, Users Group newsletters, software etc.

TECHNICAL BULLETINS FOR VZ COMPUTERS

# 88	Printing out System-80 screen graphics.	(2)
# 91	Programming the VZ-200 computer's joysticks.	(3)
# 92	Finding where variables are stored by the VZ-200's BASIC.	(3)
# 93	Problems with the X-7208 printer/plotter and Microsoft BASIC.	(1)
# 94	Using the X-3245 TP-40 printer/plotter with the VZ-200 & System-80.	(1)
# 98	Printing lower case and control characters on the VZ200/300.	(1)
#111	VZ-300 Mailing List tape to disk file conversions.	(1)
#114	Obtaining colour on the VZ300.	(1)
#116	Fixing the printer bug in the VZ Editor-Assembler.	(1)
	Letter on tapes and keyboard	(1)
	General hints on VZ	(1)
	Service Manual for printer interface	(7)
	Service Manual for disk drive controller	(12)

PRINTING OUT SYSTEM-80 SCREEN GRAPHICS

USING THE X-3252 OR X-3250 SEIKO PRINTERS

Quite a few customers with a System-80 computer and either the X-3252 (GP-80) or X-3250 (GP-100) printer have asked if there is any easy way to print out a screen of graphics characters.

For those people the following program should be of interest. Probably the easiest way to use it would be to tack it onto the end of your main program as a subroutine, and arrange to call it immediately after putting the desired graphics on the screen.

The sample printout shown was produced by adding the program in this way to the program on the System-80 Demo Tape which draws the Dick Smith logo on the screen.

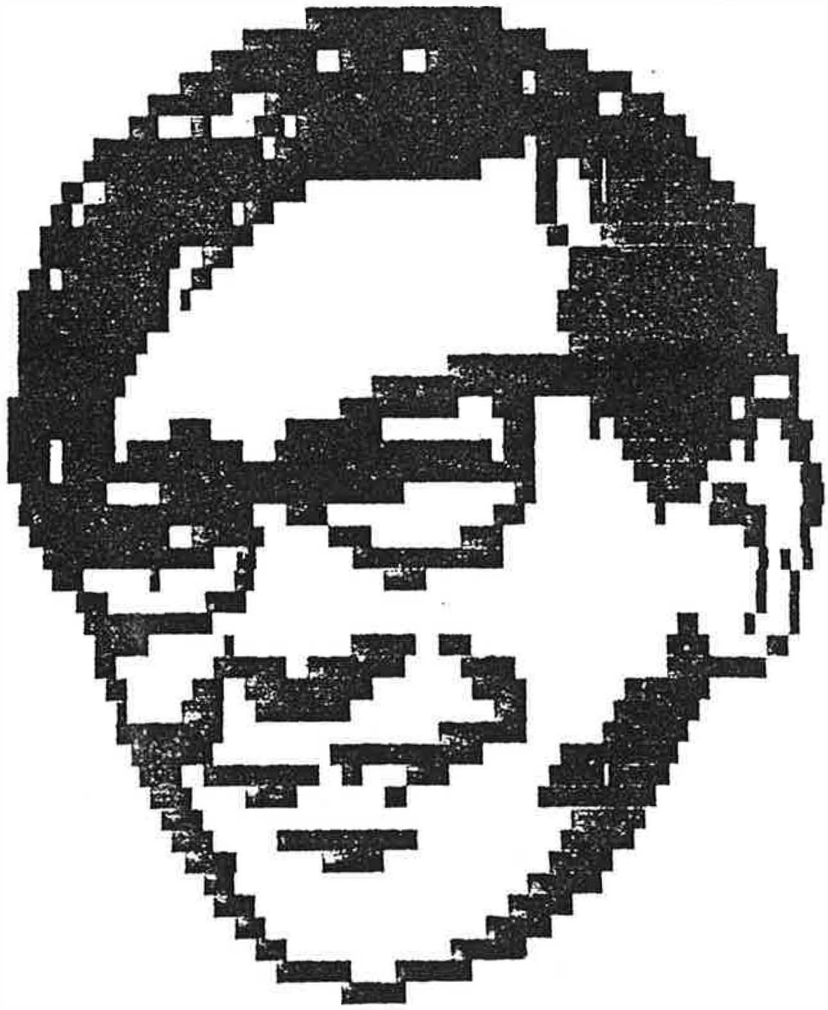
```
1000 LPRINTCHR$(8)
1010 FORB=0TO47
1020 FORA=20TO100
1030 X=POINT(A,B)
1040 GOSUB1080
1050 NEXTA
1055 LPRINTCHR$(13);
1060 NEXTB
1070 END
1080 IFX=0THEN1130
1090 FORT=1TO4
1100 LPRINTCHR$(255);
1110 NEXTT
1120 RETURN
1130 FORT=1TO4
1140 LPRINTCHR$(128);
1150 NEXTT
1160 RETURN
```

This program was written for a GP-80, which cannot handle the full width of the screen print because of its 80-column format. This is the reason for the limited range for A in line 1020. The actual start and finish values for A can be changed to vary the part of the screen that is printed.

Regards,

B. Whipps

Bernard Whipps,
Service Department



As you can see, the program prints out the screen with quite accurate proportions.

Dick Smith Electronics

Technical Bulletin

PROGRAMMING FOR THE VZ-200 COMPUTER'S JOYSTICKS

The VZ-200's optional joysticks are interfaced and software scanned in a similar fashion to the main keyboard. The interface occupies port addresses 20H to 2FH, and the joystick switches are connected in an array whose row lines are connected to port address lines A0 to A3. This effectively places the switches at the following bit positions and addresses:

		BIT POSITION 4	3	2	1	0	
PORT ADDRESS	2EH (46D)	ARM	→	←	↓	↑	RIGHT JOYSTICK
	2DH (45D)	FIRE					RIGHT JOYSTICK
	2BH (43D)	ARM	→	←	↓	↑	LEFT JOYSTICK
	27H (39D)	FIRE					LEFT JOYSTICK
	20H (32D)	TEST OVERALL JOYSTICK STATUS					

Note that the port addresses shown above are those which cause the joystick row concerned to go low -- i.e., to logic 0. The first four addresses cause only the row concerned to go low, to test just that row, while the fifth address pulls all four rows low simultaneously, to allow a quick check of overall joystick status. In each case, if a joystick is moved from its rest position, one or more bit lines will be pulled low (logic 0). If the joystick is in its rest position, all bit lines will remain high (logic 1).

The status of the joysticks is easy to determine from within a program, both in BASIC and assembly language. In BASIC the easiest way is to use the INP command with AND 31 (or AND 16) to mask off the unused data bits, then testing for the bit(s) pulled low, as shown by the first example:

.../2

PROGRAMMING FOR THE VZ-200 JOYSTICKS, PAGE 2:

```
5  R$="RIGHT JOYSTICK: ":L$="LEFT JOYSTICK: "
10 A=INP(32)AND31:IFA=31THEN10:REM WAIT FOR SOME ACTION
20 A=INP(46)AND31:IFA=31THEN100:REM CHECK FIRST ROW
30 IFA=26THEN PRINT R$+"LEFT+UP":GOTO200
32 IFA=25THEN PRINT R$+"LEFT+DOWN":GOTO200
34 IFA=22THEN PRINT R$+"RIGHT+UP":GOTO200
36 IFA=21THEN PRINT R$+"RIGHT+DOWN":GOTO200
40 IFA=30THEN PRINT R$+"UP":GOTO200
50 IFA=29THEN PRINT R$+"DOWN":GOTO200
60 IFA=27THEN PRINT R$+"LEFT":GOTO200
70 IFA=23THEN PRINT R$+"RIGHT":GOTO200
80 IFA=15THEN PRINT R$+"ARM":GOTO200
100 A=INP(45)AND16: REM NOW CHECK SECOND ROW
110 IFA=0THEN PRINT R$+"FIRE":GOTO200
120 A=INP(43)AND31:IFA=31THEN190:REM CHECK 3RD ROW
130 IFA=26THEN PRINT L$+"LEFT+UP":GOTO200
132 IFA=25THEN PRINT L$+"LEFT+DOWN":GOTO200
134 IFA=22THEN PRINT L$+"RIGHT+UP":GOTO200
136 IFA=21THEN PRINT L$+"RIGHT+DOWN":GOTO200
140 IFA=30THEN PRINT L$+"UP":GOTO200
150 IFA=29THEN PRINT L$+"DOWN":GOTO200
160 IFA=27THEN PRINT L$+"LEFT":GOTO200
170 IFA=23THEN PRINT L$+"RIGHT":GOTO200
180 IFA=15THEN PRINT L$+"ARM":GOTO200
190 A=INP(39)AND16: REM CHECK FOURTH ROW
195 IFA=0THEN PRINT L$+"FIRE"
200 FOR I=1TO300:NEXTI:GOTO10
```

In assembly language programs it is even easier to read the joystick status. Here is a sample subroutine which reads the status of both joysticks and returns with the results in the B and C registers. Note that in each case the appropriate bit is set to logic 1 if that joystick switch is enabled, except that the two 'FIRE' switches are transferred to bit 5.

I.e., the bit assignment becomes:

BIT	5	4	3	2	1	0
SWITCH	FIRE	ARM	→	←	↓	↑

.../3

PROGRAMMING FOR THE VZ-200 JOYSTICKS -- PAGE 3:

```
JOYSTK  IN   A,(2EH)      ;read 1st row
        OR   0E0H         ;set bits 5-7 high
        CPL               ;then complement to invert
        LD   B,A          ;& save in B reg
        IN   A,(2DH)      ;read 2nd row
        BIT  4,A          ;check for FIRE pressed
        JR   NZ,JOYST1    ;skip if not
        SET  5,B          ;otherwise set bit 5
JOYST1  IN   A,(2BH)      ;read 3rd row
        OR   0E0H         ;& process as above
        CPL
        LD   C,A          ;except save in C reg
        IN   A,(27H)      ;read 4th row
        BIT  4,A          ;check for FIRE pressed
        RET  NZ           ;return if not
        SET  5,C          ;otherwise set bit 5
        RET               ;& then leave
```

I hope this information is enough to allow you to program the joysticks with confidence.

Regards,

Jamieson Rowe

....
Jamieson Rowe,
Technical Director

DICK SMITH ELECTRONICS

PERSONAL COLOUR COMPUTER

JOYSTICK × 2

with interface

X-7315

INSTRUCTION MANUAL

MADE IN HONG KONG

91-0159-22

These fast-response Joysticks offer you 8-direction flexibility and both an ARM and FIRE control button. The Interface allows your computer to support these Joysticks.

CAUTION!

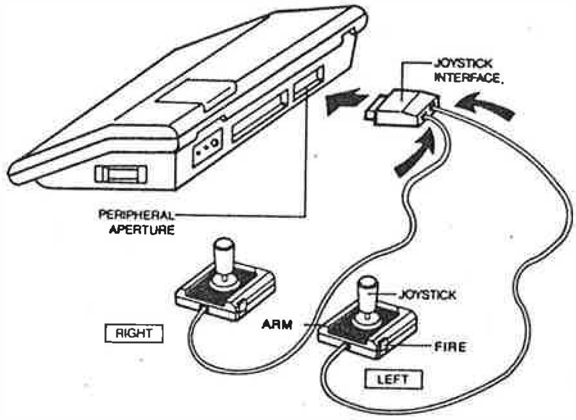
1) Disconnect all power to computer before connecting any expansion module.
2) Keep expansion sockets of both computer and expansion moudles clean and free of liquids of any kind.

NOTE: FAILURE TO FOLLOW THESE PRECAUTIONARY STEPS MAY CAUSE IRREPARABLE DAMAGE TO YOUR EQUIPMENT.

INSTALLATION

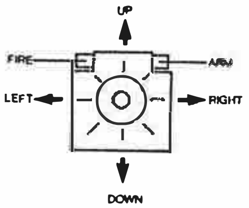
1) Check to be sure that the power is off.
2) Turn to the back panel of the computer.
3) Remove the "PERIPHERAL" cover by taking out the screws.
4) PLUG the Joystick Interface into the "PERIPHERAL" socket slowly and smoothly. Check to be sure the Interface is fully inserted and firmly attached.
5) Turn on the power to the computer and check that computer operation has not changed. (For correct procedure, refer to your computer User's Manal)

If the TV screen does not display the "READY" message, turn off the power, remove the interface unit from the computer, re-insert, slowly and smoothly. Then turn on the power and go through the computer User Manual procedure.



JOYSTICK

There are totally 8 directions on your joystick together with the Arm and Fire buttons. You may feel free to move the joystick to any of these 8 directions.



TECHNICAL INFORMATION

BASIC access

Using the BASIC language, you can access the left and right Joysticks by using the A=(INP(N)AND 31) command. The following tables give the various values of A and N for different directions or buttons.

NOTE: L=left/R=right/U=up/D=down

LEFT JOYSTICK

N		A	DIRECTIONS OR BUTTONS
43		26	L&U
43		25	L&D
43		22	R&U
43		21	R&D
43		30	U
43		29	D
43		27	L
43		23	R
43		15	FIRE
29			ARM

15

RIGHT JOYSTICK

N		A	DIRECTIONS OR BUTTONS
46		26	L&U
46		25	L&D
46		22	R&U
46		21	R&D
46		30	U
46		29	D
46		27	L
46		23	R
46		15	FIRE
46			ARM

15

By checking the value of A, you can determine the status of the left or right Joystick. For example, the following BASIC program will check the left, right, up and down directions of the LEFT Joystick.

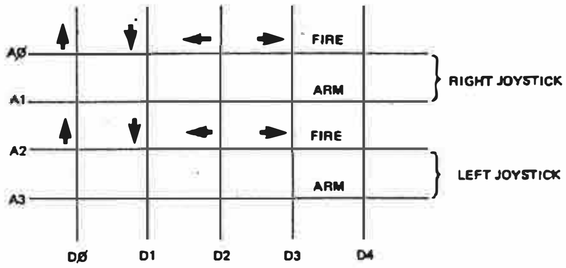
```
Example:   1Ø CLS
           2ØA = (INP (43) AND 31)
           3Ø IF A = 30 THEN PRINT "UP": GO TO 2Ø
           4Ø IF A =29 THEN PRINT "DOWN": GO TO 2Ø
           5Ø IF A= 27 THEN PRINT "LEFT": GO TO 2Ø
           6Ø IFA =23 THEN PRINT "RIGHT"
           7Ø GO TO 2Ø
```

When this program is running, you can move the LEFT Joystick in any direction, and that direction will appear on screen.

To stop this program, press the CTRL and – keys at the same time to get BREAK

Assembly Language access

In assembly language, you can access the right or left Joystick by utilizing the following Joystick matrix.



Use the IN Instruction to read the Joystick whose address ranges from hexadecimal '20 to 2F. You can write a program to scan the address lines and check which data bit has become 0.

Dick Smith Electronics

Technical Bulletin

FINDING WHERE VARIABLES ARE STORED BY THE VZ-200'S BASIC

When programming the VZ-200 computer in BASIC, there are times when you need to know where the interpreter has stored your program's variables. Superficially this is not easy, as the VZ-200's BASIC seems to have no VARPTR function. However as it happens the VARPTR routine is actually present inside the VZ-200's ROMs, even though the input/tokenising section of the interpreter cannot recognise the keyword 'VARPTR' and turn it into the appropriate token.

To use the routine simply, it is possible to 'trick' the interpreter by POKEing the appropriate token (C0 hex or 192 decimal) into a program line, in place of a similar token -- say that for USR (C1 hex or 193 decimal). As the execute section of the interpreter can treat the VARPTR token normally and call the appropriate routines, this gives the desired result when the program is RUN. However because the LIST and LLIST routines cannot recognise the VARPTR token, the line with this token in cannot be listed properly.

Here is a small sample program which should illustrate how the above 'trickery' can be performed from within your BASIC program itself -- in this case for a string variable:

```
 10 GOTO30
 20 X=USR(A$):RETURN
 30 D=PEEK(30884)+256*PEEK(30885):REM FINDS START OF PROG
 40 B=PEEK(D):IFB<>193THEND=D+1:GOTO40:REM FIND 'USR' TOKEN
 50 IFB=193THENB=192:POKED,B:REM & REPLACE WITH 'VARPTR' TKN
 60 REM NOW TRY IT OUT
 70 A$="WHATEVER":GOSUB20:REM GO FIND PTR FOR A$
 80 FOR I=XTOX+2:LPRINTI,PEEK(I):NEXTI:REM PRINT PTRS OUT
 90 PT=PEEK(X+1)+256*PEEK(X+2):REM NOW SET PT FOR A$ START
100 FOR I=PT TO PT+PEEK(X)-1
110 LPRINTCHR$(PEEK(I));
120 NEXTI:LPRINT
```

The actual line which eventually calls the VARPTR routine is line 20, which is placed as near as possible to the start of the program so that it can be located easily to swap tokens.

.../2

FINDING WHERE VARIABLES ARE STORED BY THE VZ-200'S BASIC -- 2

As you can see it is made a subroutine, so that the VARPTR function can effectively be called from anywhere in the program using GOSUB20. Line 10 is simply arranged to skip over the subroutine to line 30, the 'real' start of the program proper.

Lines 30 to 50 perform the actual token swapping in line 20. First line 30 finds the pointer to the start of BASIC's program storage area. Then line 40 examines each byte in memory, starting at the beginning of the program, until it finds the USR token (decimal 193). Then line 50 pokes the VARPTR token code back into the same address. So after these lines are RUN, line 20 will behave as if it were written:

```
20 X=VARPTR(A$):RETURN
```

Lines 70 to 120 are to demonstrate how the subroutine works. First, line 70 gives string variable A$ a value, then calls the subroutine so that X will be given the pointer value for it. Then line 80 prints out the string's length and storage address bytes. Finally line 90 sets PT to point to the actual string storage address, and lines 100-120 read it and print it out.

If you RUN this program, this is what you get:

```
 31701          8
 31702          86
 31703          123
WHATEVER
```

As you can see, the first byte stored at the pointer address for a string variable is the length of the string. The next two bytes form the pointer to where the string is actually stored.

Note that if you try to LIST or LLIST the above program after it has been run, line 20 will look like this:

```
20 X=_
```

As noted earlier, this is simply because the listing routines cannot identify the VARPTR token.

.../3

FINDING WHERE VZ-200'S BASIC STORES ITS VARIABLES -- 3

Needless to say, this approach isn't confined to finding string variables. It can be used for each type of variable, although line 20 will obviously need to be changed to suit the type of variable involved. For example if you use X=USR(A) it would be suitable for either integer or single-precision numeric variables.

Here is the format used for the pointers retrieved for the various types of variable (X = first pointer location):

```
INTEGER VARIABLE:   X   contains LSB of variable itself
                    X+1 contains MSB

SINGLE PREC. VBL:   X   contains LSB of variable itself
                    X+1 contains next most sign. byte
                    X+2 contains MSB
                    X+3 contains exponent of value

DOUBLE PREC. VBL:   X   contains LSB of variable itself
                    X+1 contains next most sign. byte
                    X+2 contains next most sign. byte
                    X+3 contains MSB
                    X+4 contains exponent of value

STRING VARIABLE:    X   contains length of string
                    X+1 contains LSB of string's start
                    X+2 contains MSB of string's start
```

By modifying the above technique slightly, you could have two or more 'VARPTR' routines, one to suit each type of variable your program needs to find.

Regards,

Jamieson Rowe,
Technical Director

Dick Smith Electronics

Technical Bulletin

PROBLEMS WITH THE X-7208 PRINTER/PLOTTER AND MICROSOFT BASIC

Some users of our printer/plotter have noticed that when they 'LPRINT' the 'LINE-UP' code (0B hex or 11 decimal) the printer seems to perform continual line feeds. In fact the printer is operating correctly and the computer is interpreting the 'LINE-UP' code as an 'unconditional skip to the top of the next page' and as such sends line feeds to the printer. It should be noticed that this will not occur with all computers, but only those running Microsoft or similar basics that perform in-line filtering for certain codes i.e. form feeds etc.

Luckily there is an easy solution to this problem. For those codes that seem to cause problems with your basic (0BH for the SYSTEM 80 or VZ-200) simply use the 'OUT' command instead of LPRINT.

e.g. for the SYSTEM 80   OUT 243,11

for the VZ-200   OUT 1,11

This will perform exactly the same function as the 'LPRINT' statement but it by-passes the computer's 'filtering' of the printer output.

Regards,

Steven Engel.

Dick Smith Electronics

Technical Bulletin

Using the X-3245 TP-40 printer plotter with the VZ-200 and System 80

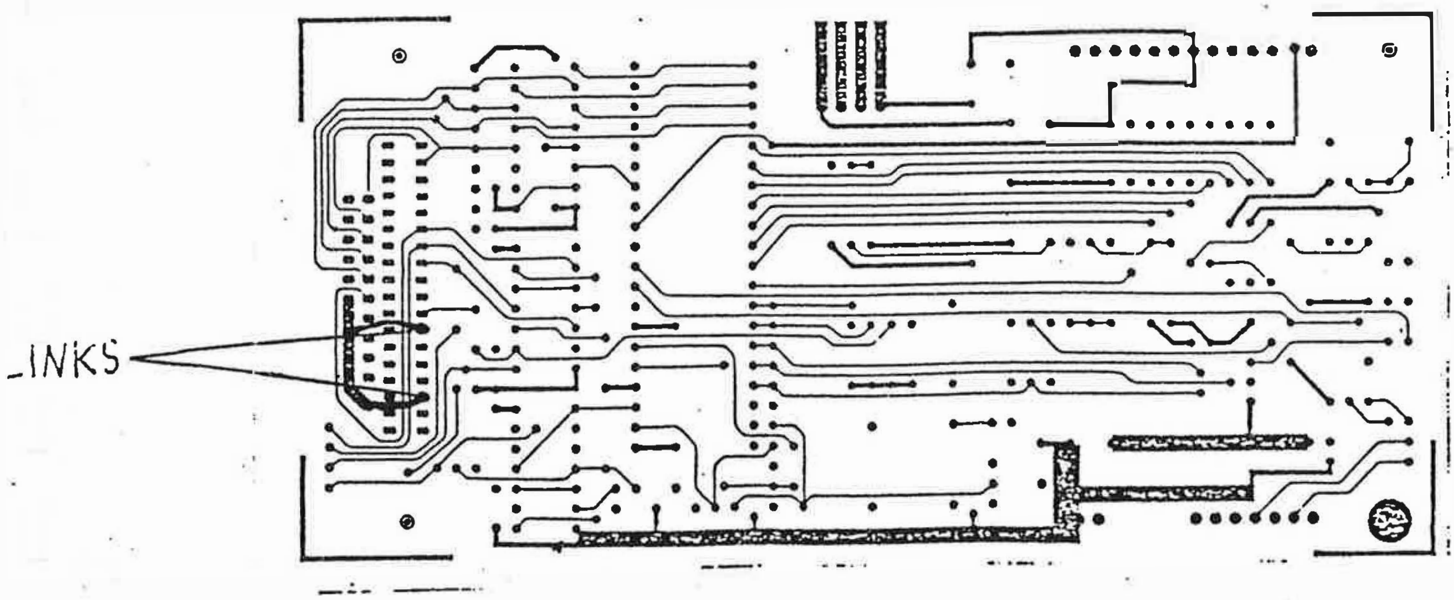
The TP-40 will not work with the VZ-200 because there is no ground connection between the two. The VZ-200 uses pin 16 of the 36way connector as the ground rather that one of the standard pins.

The TP-40 will not work with the System 80 because the "paper empty" signal on pin 12 of the 36way connector floats high, indicating no paper.

The solution is simple, open the TP-40 and remove the two printed circut boards. Link pins 12 and 16 of the 36way connector to earth using 2 links on the underside of the PCB as shown in the diagram below.

Regards,

Paul Beaver
Computer Products Co-ordinator



Cnr. Lane Cove & Waterloo Rds, North Ryde, NSW 2113 Ph (02) 888 3200

No 98

Dick Smith Electronics

Technical Bulletin

Printing Lower Case and Control Characters on the VZ-200/300

As you may be aware the VZ-200/300 computers do not display the standard lower case ASCII characters. Instead they display inverse and graphics characters. To obtain a printout of these special characters the BASIC switches the printer to the graphics mode, outputs a graphic image of the non-standard character. Then switches the printer back to the text mode (NOTE. the X-3250 GP-100 printer is the only printer that the special characters can be printed on).

Because the BASIC used in the VZ-200/300 filters certain ASCII characters when the LPRINT or LLIST statements are used when you wish to print one of these characters in their standard form an OUT statement must be used. Here is a program lising to show the use of the OUT statement.

```
10 FOR X= 1 33 TO 127
20 OUT 1,X
30 OUT 2,X
40 FOR T= 1 TO 100:NEXT T
50 NEXT X
60 OUT 1,0
70 OUT 2,0
```

There are three points to notice in the above program

1. You must OUT to both ports 1 and 2 in that order.

2. Because the OUT statement does not check the status of the port you must include a delay loop (see line 40) to allow the printer time to actually print each character.

3. Because the OUT statement can disturb the BASIC stack pointers you must reset the parallel port before using the LPRINT or LLIST statements. This is done with the lines 60 & 70.

It is important to know that the VZ-200/300 computers use the ASCII character 00 (NUL) to reset the parallel port. This means that printers that use the ASCII 00 as a control character will have some problems in operation on the VZ-200/300 computers. In most cases (but not in all) printers that use the ASCII 00 have an alternate character that can be used (usually ASCII 128).

P/S Beaver

Paul Beaver

Cnr Lane Cove & Waterloo Rds, North Ryde, NSW 2113 Ph (02) 888 3200

Dick Smith Electronics

Technical Bulletin

VZ-300 MAILING LIST TAPE TO DISK FILE CONVERSIONS

Below are the changes to be done to the B.A.S.I.C program to allow files to be saved on disk instead of tape for (X-7259) Mailing List program.

Once you have loaded Mailing List BREAK the program and type in the lines below pressing (RETURN) after each line.

```
1040 PRINT@162,"2. READ DATA FROM DISK";
1080 PRINT@290,"6. WRITE DATA TO DISK ";
5020 PRINT@270,"[WRITE DATA TO DISK]"
5030
5040
5050
5060
5070
5080
5110
5120
5205 ERA"MAILDATA"
5210 OPEN "MAILDATA",1:PR#"MAILDATA",DT
5230 PR#"MAILDATA",D$(N)
5240 NEXT:CLOSE"MAILDATA"
6020 PRINT@70,"[ READ DATA FROM DISK ]";
6030
6040
6050
6060
6070
6080
6100 OPEN "MAILDATA",0:IN#"MAILDATA",DT:IF DT=0THEN6135
6120 IN#"MAILDATA",D$(N)
6135 CLOSE"MAILDATA"
7030 PRINT@199,[FUNCTION COMPLETE];
7050 SOUND 30,2:RETURN
```

Now SAVE"MAILLIST" to disk.
Type NEW
RUN below program .

```
10 OPEN"MAILDATA",1
20 PR#"MAILDATA",0
30 CLOSE"MAILDATA"
```

The above program has prepared the disk with the MAILLIST program to save and read files. The above program will never be used again.

Now you have finished just RUN"MAILLIST" and the instructions are as per old Mailing List program. The only difference is that it saves and loads files a lot faster.

Compiled by Jamie PERRY J Perry

Cnr. Lane Cove & Waterloo Rds. North Ryde, NSW, 2113. Ph. (02) 888 3200

Dick Smith Electronics

Technical Bulletin

OBTAINING COLOUR ON THE VZ300

1. Background colour - this can be either green or buff (pink), and is changed by entering:

```
     COLOR, 0 (for green - default color)
or   COLOR, 1 (for buff)
```

This leaves the foreground colour of graphics characters unchanged. Note also that only these graphics characters which are to be found on the following keys, (while pressing the shift key: Q, W, E, R, T, Y, U, I, A, S, D, F, G, H, J) may be used to change foreground colours. Any others will always remain black.

2. Foreground colour - this can be changed in two ways.

a) Changing the background colour as well:
enter - COLOR I, J
where I is the foreground colour from 1 to 8 (list of colour codes below) and J is the background colour as above.

b) Changing only the foregound colour:
enter - COLOR I
where I ranges from 1 to 8 as in a).

Note: In both a) and b), no changes to foreground colour will be noticed unless followed by a print statement.

COLOUR CODE TABLE

Code	Colour
1	Green
2	Yellow
3	Blue
4	Red
5	Buff (pink)
6	Cyan
7	Magenta
8	Orange

eg:

```
 10   for I = 1 to 8
 20   Color I
 30   Print '<shift QWE>'
 40   Next I
 50   Color, 1
 60   For I = 1 to 8
 70   Color I
 80   Print '<shift UIA>'
 90   Next I
100   Color 2, 1
110   Print '<shift GHJ>'
```

Regards -
David Summers

Cnr. Lane Cove & Waterloo Rds, North Ryde, NSW, 2113. Ph. (02) 888 3200

No. 116

Dick Smith Electronics

Technical Bulletin

Fixing the printer bug in the VZ-editor assembler.

Below is a patch to enable your editor assembler to list its source code. As stated in the manual using option C.

First enter Insert mode by entering 'I'. Then set code origin by entering 'O'. Now type in the below program, pressing RETURN at the end of each line.

```
         LD   BC,0CH       ;Size of transfer is 12 bytes.
         LD   HL,LOOP      ;Point to new printer routine
         LD   DE,8F54H     ;Point to editor assembler print out
         LDIR              ;Transfer routine to editor assembler
         JP   7B00H        ;Return control to editor assembler
    LOOP IN A,(00H)        ;Load printer status
         BIT 0,A           ;Check ready bit
         JR NZ,LOOP        ;Repeat LOOP if not ready
         LD A,C            ;Load Accumalator with print data
         OUT (0EH),A       ;Output data to printer port
         OUT (0DH),A       ;Another port for an early interface
         RET               ;Get next character
```

Now assemble the program by entering 'A'. Now RUN the program by entering 'R' then press 'Y' to verify you wish to execute the program. Finish up by deleteing the program by entering 'D*'. Your editor assembler may list programs now, just by selecting option 'C'.(enter 'SC').

Yours sincerly

Jamie PERRY

64K RAM
MEMORY EXPANSION MODULE

Installation Manual

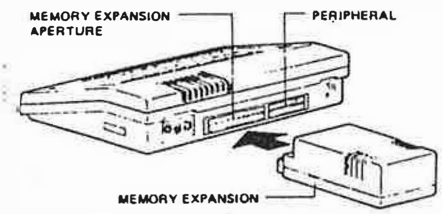
X-7306

MADE IN HONG KONG 91-0165-12

MEMORY EXPANSION MODULE

The 64K Memory Expansion Module can provide you plenty of memory space.

INSTALLATION



WARNING: a) TURN OFF THE POWER TO YOUR COMPUTER BEFORE CONNECTING ANY EXPANSION MODULE
b) KEEP THE SLOT OF THE COMPUTER AND THAT OF THE EXPANSION MODULE CLEAN AND AWAY FROM LIQUIDS

1) Turn the computer to its back side.
2) Remove the cover labelled as "MEMORY EXPANSION" by unscrewing it.
3) Plug the expansion module into the slot on the computer slowly and smoothly. Check that the module is fully inserted and firmly attached to the computer.
4) Turn on the power to your computer and check that the computer works as before. (Please refer to the User Manual) If the TV screen does not display the message "READY", TURN OFF the POWER. Detach the expansion module from the computer slowly and smoothly and repeat procedure 3) again.
5) Try the following instructions to check whether the expansion module is properly inserted.

   TYPE IN

   PRINT PEEK (3Ø897) RETURN

   PRINT PEEK (3Ø898) RETURN

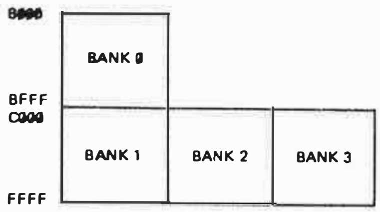
   RETURN – press the key labelled as RETURN

   For the first instruction, the result will be 255
   For the second instruction, the result will be 255.
   If the result does not match with that listed TURN OFF the POWER.
   Detach the expansion module from the computer smoothly slowly and repeat procedure 3) again.
   If the result matches with that listed the set-up procedure is all-right.

HOW TO FULLY USE THE 64K MEMORY?

The 64K Memory Expansion Module is arranged in four 16K-memory banks. The first bank locates in 8ØØØH* to BFFFH* and is always present. The remaining three 16K-memory banks are located in CØØØ H* to FFFF H* and are selected by a software switch.

*H represents hexadecimal values



The software switch is a write only latch and is located in the I/O port address 7FH. The codes to select the memory banks are listed as below:

Code to Select memory bank	memory bank
Ø or 1	BANK 1
2	BANK 2
3	BANK 3

WARNING: do not switch the memory bank in BASIC.

A user writing in BASIC can only access 32K RAM. It is because the stack of the system is located near the top of the memory, i.e. inside the switchable memory bank. If the user tries to switch the memory bank, the system will crash.

The bank switching feature offers a user writing in assembly language 32K more RAM space. However, the user should assign the stack pointer and his main program in BANK Ø. The subroutines and data should be assigned in BANK 1, 2 or 3. The user must keep track of the bank position of the subroutines and data. The main program should select the appropriate memory bank before calling the subroutines or accessing the data in the switchable bank.

Dick Smith Electronics Pty Ltd INCORPORATED IN NSW

Head Office: Cnr Lane Cove Rd and Waterloo Rd. North Ryde. NSW 2113, Australia
Postal Address: P.O.Box 321, North Ryde NSW 2113, Australia
Telephone: (02) 888 3200 **Telex:** AA20036 **Cables:** DIKSMIT Sydney
International Phone No.: 61 2 888 3200

The Australian Company

Dear customer,

We have had a few reports of problems with the VZ-200 computer and hope that these notes will help explain them.

1) Tape loading problems.

Some of these have been due to faulty demonstration tapes, but most appear to have been caused by some brands of recorder not operating correctly on playback when they are connected to an external load of more than 15 ohms. At present the VZ-200 cassette input circuit provides 470 ohms. If you are having cassette loading troubles a solution is to make up an adaptor lead with a 3.5mm jack (P-1231), a 3.5mm plug (P-1132) and a 15 ohm resistor (R-1030) all wired in parallel.

2) Keyboard problems.

A number of people have reported what they believe to be 'BUGS' in the VZ-200 computer, with regard to the 'one-key command entry' feature. One of these is that when you tried to get some of the commands shown 'underneath' the keys, the computer immediately gave you a whole string of other commands as well.

Basically this is not a bug, but due to the VZ-200 manual being misleading in its instructions on how to get the 'under key' commands. You don't hold down <u>both</u> the CNTL and RETURN keys before pressing the third key - rather, you hold down only the CNTL key, and then first press the RETURN key, and then the third key. If you do this, the above trouble won't occur.

The other main complaint is that while the 'auto-repeat' works on the one-key commands which are <u>above</u> the keys, it doesn't on those below. This is a minor 'bug' but not the one most people think. The one-key commands <u>shouldn't</u> repeat at all, because they are only needed singly. So the minor bug is that the ones above the keys <u>do</u> repeat, not that those below <u>don't</u>.

Regards,

<u>DICK SMITH ELECTRONICS SERVICE DEPARTMENT.</u>

DARRELL'S COMPUTER TIPS : Please Take One

Tapes :

- Always make a Back-Up copy of your program - FIRST
- Always name your programs when saving them.
- Always verify a newly saved program.
- Always save to an erased section of tape.
- Use good quality tapes - preferably computer cassettes.
- Don't use tapes longer than 60 minutes.
- Clean your cassette heads regularly.
- Demagnetise your heads regularly.
- Don't save your programs on the plastic tape leader.
- Don't press the 'Record' button when loading programs.

Use a suitable tape recorder. Manufacturers' have different standards for head gap size, azimuth alignment, ALC attack & depth etc. and these variables and others determine which recorders will work best with a particular computer.

Programs from books - watch out for :

- \- I - Don t confuse with - 1 -
- \- O - Don't confuse with - 0 -
- \- : - Don't confuse with - ; -
- \- . - Don't confuse with - , -
- \- ' - Don't confuse with - " -

When running a program from a book for the first time you'll find (if you're a typist like me) a number of errors caused by typing mistakes. If after fixing them the program still doesn't work correctly there may be some commands used in the program that your computer doesn't use, or uses differently to the computer the program was written on. If this is so then the book by David Lien 'The Basic Handbook' will be invaluable. It shows the commands of all the popular computers, and ways to emulate those commands your computer may not have.

Hardware :

- make sure that all leads are plugged into the correct sockets.
- Make sure that everything is turned on.

Debugging programs :

Your best tools for this are 'TRON', 'STOP' & the Snapshot. Use 'TRON' to find if the program is going to the lines it is supposed to or not. Use 'STOP' at critical points in the program, print out the values of suspect variables 'PRINT A,B,C' and then use 'CONT' to go on until you find the problem. The Snapshot: In a program you have three suspect variables e.g. A,B,C. Insert at appropriate places in the program the following: PRINT "A=";A;"B=";B;"C=";C This will display the values of these variables as the program runs. You can also send these values to the printer so as not to spoil the screen display.

SERVICING MANUAL FOR LASER 200 PRINTER INTERFACE

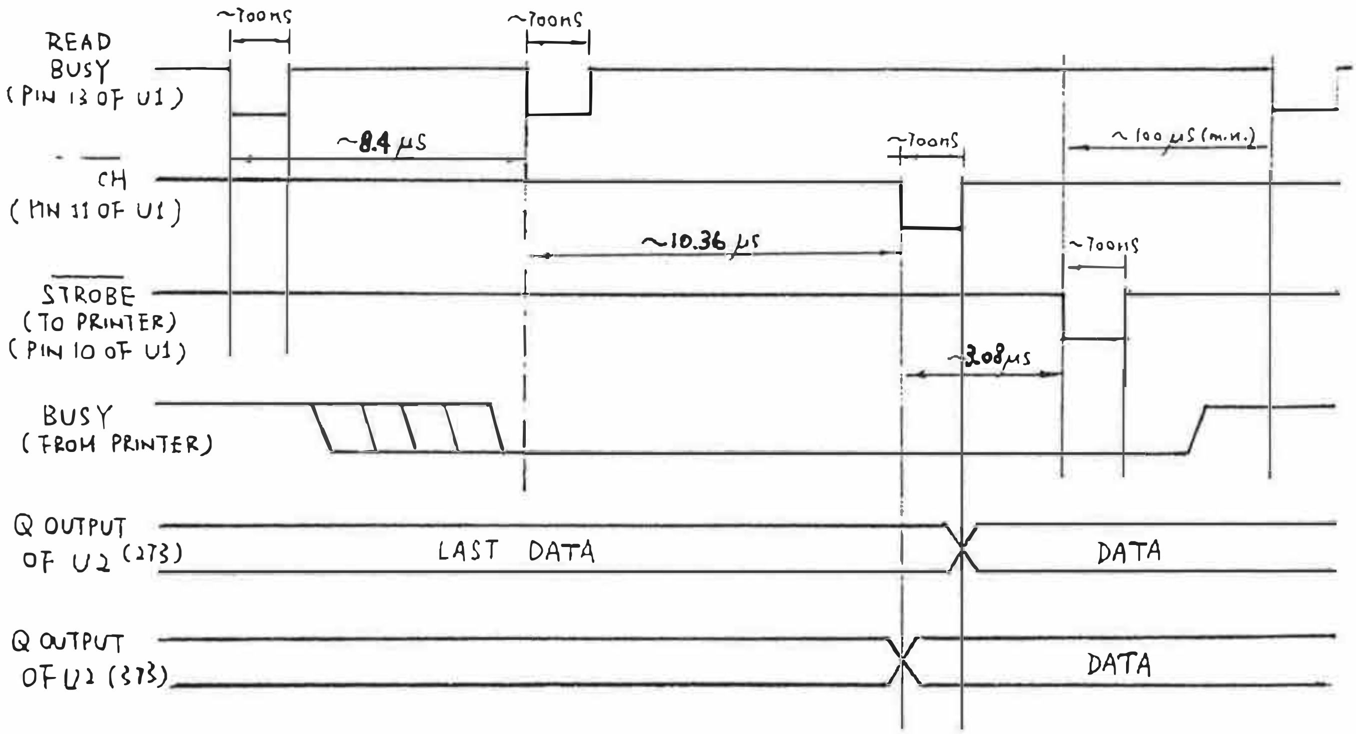
I. ELECTRICAL SPECIFICATION

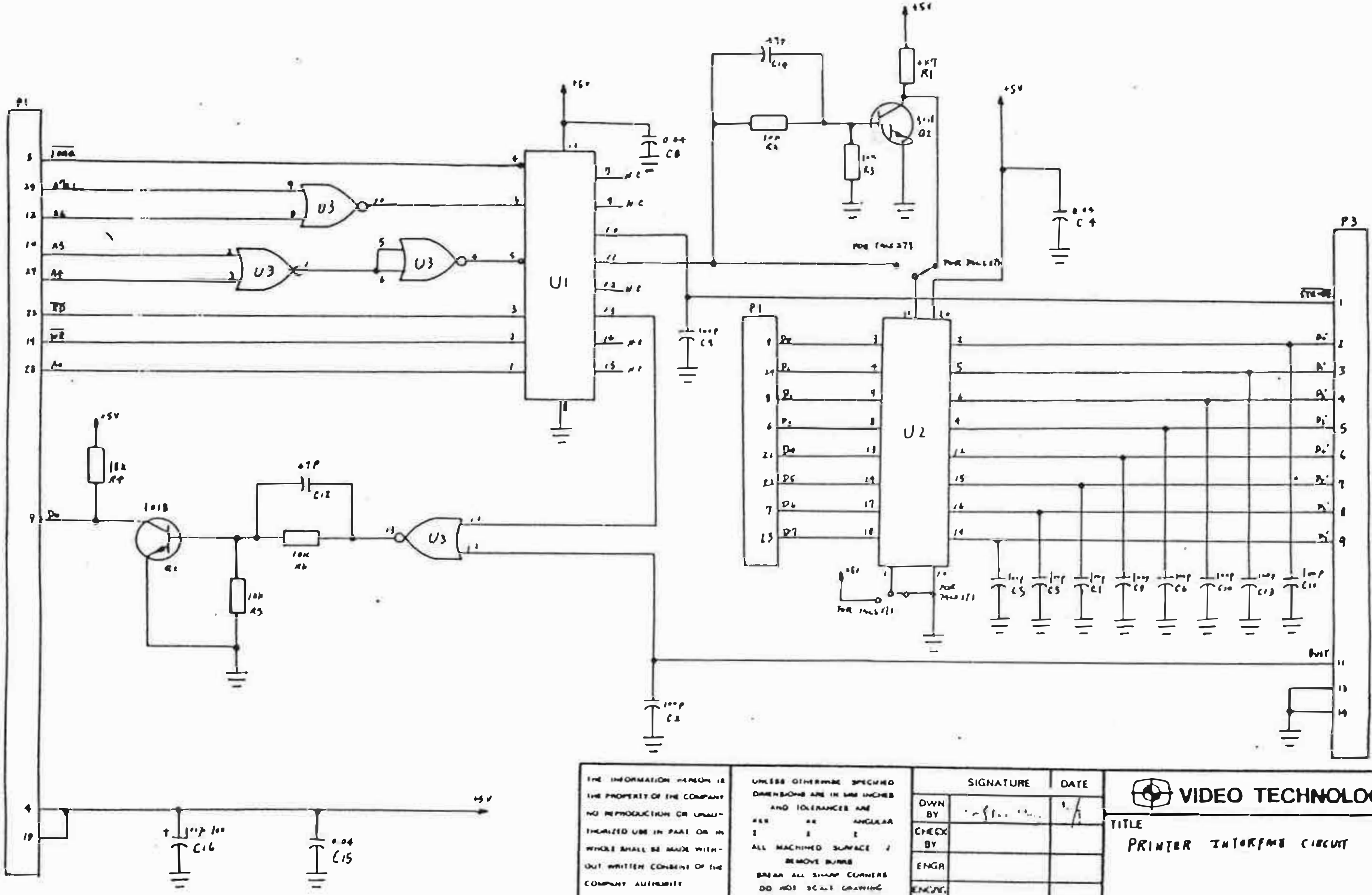
Interface standard : Centronics Bus Interface

Supply voltage : Single +5V DC

Current consumption : 50mA (max)

II. TIMING DIAGRAM





REVISION					
EV	ZONE	DESCRIPTION	DRAWN	APPD	DATE

THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENG'G AUTH		

NEXT ASSY

MATERIAL

VIDEO TECHNOLOGY LTD

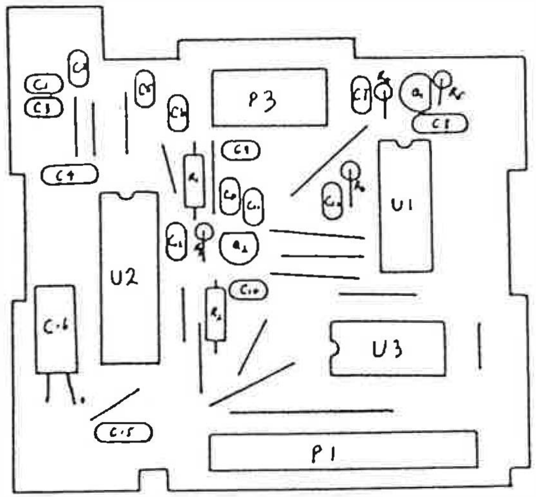
TITLE
PRINTER INTERFACE CIRCUIT

SIZE | CODE IDENT | DWG NO

4	3	2	1

REVISION

ZONE	DESCRIPTION	DRAWN	APPD	DATE



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE .XXX .XX ANGULAR ± ± ± ALL MACHINED SURFACE / REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENG'G AUTH		

VIDEO TECHNOLOGY LTD

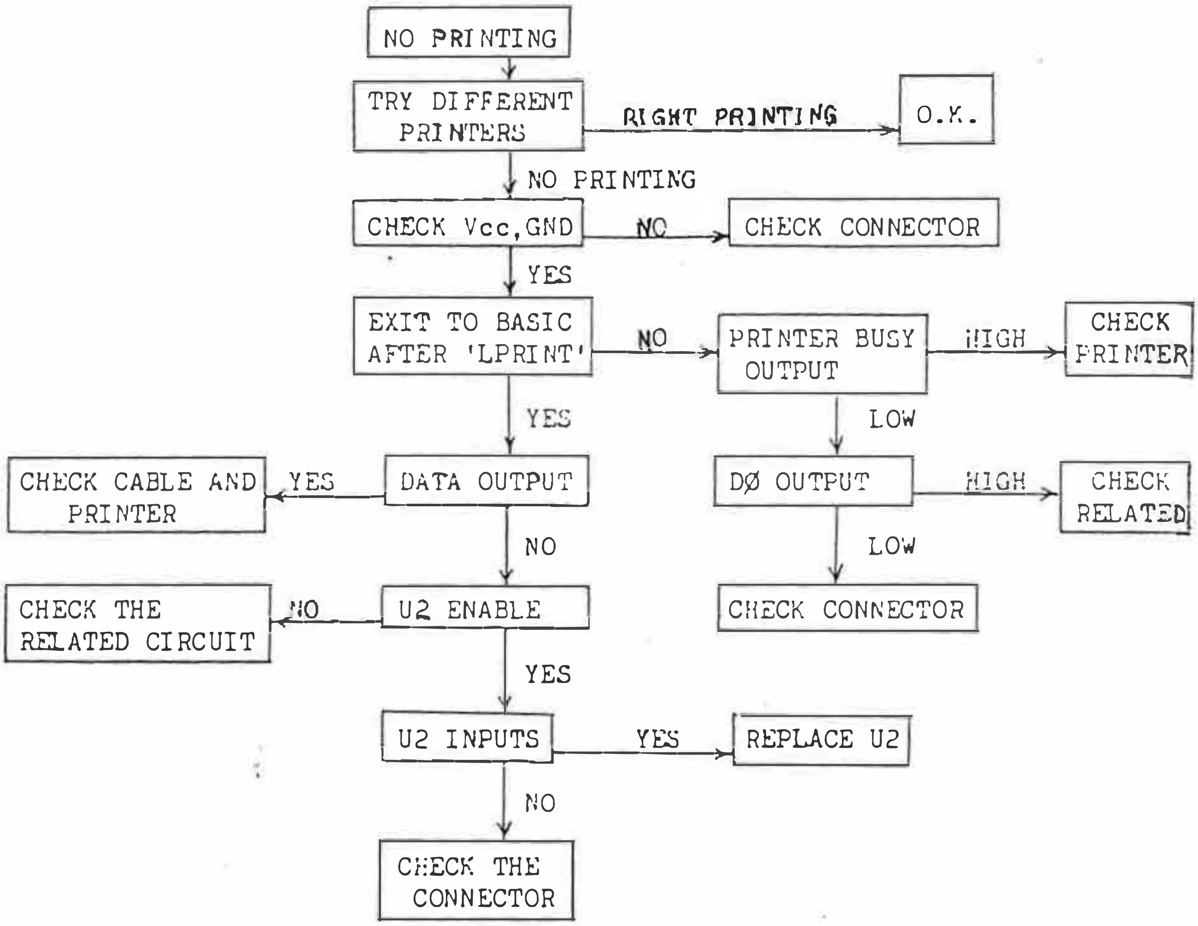
TITLE COMPONENT LAYOUT

PRINTER INTERFACE

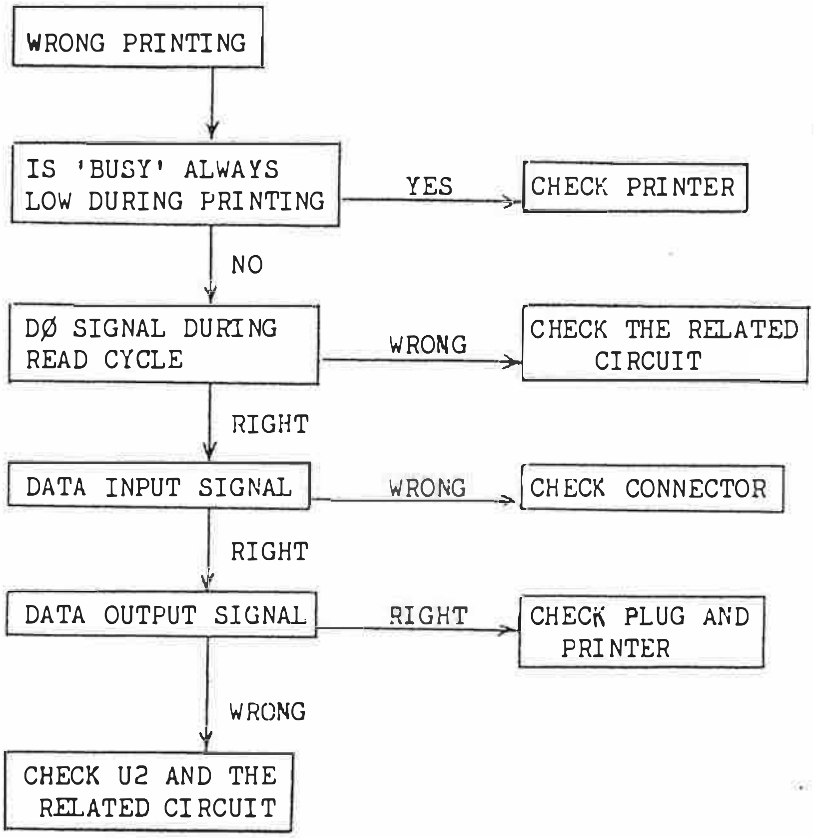
NEXT ASSY		MATERIAL	SIZE	CODE IDENT	DWG NO
USED ON		FINISH	A2		

V. TROUBLE SHOOTING GUIDE

1. NO PRINTING



2. WRONG PRINTING



VI. COMPONENT LIST

U1	74LS138
U2	74LS373/74LS273
U3	74LS02
Q1-Q2	9018
R1	4.7K ohm
R2-R6	5 X 10K ohm
C1-C3, C5-C7, C9-C11, C13, C16	100pf
C4, C8, C15	0.04uf
C12, C14	47pf

SERVICE MANUAL FOR DI-40
DISK DRIVE CONTROLLER

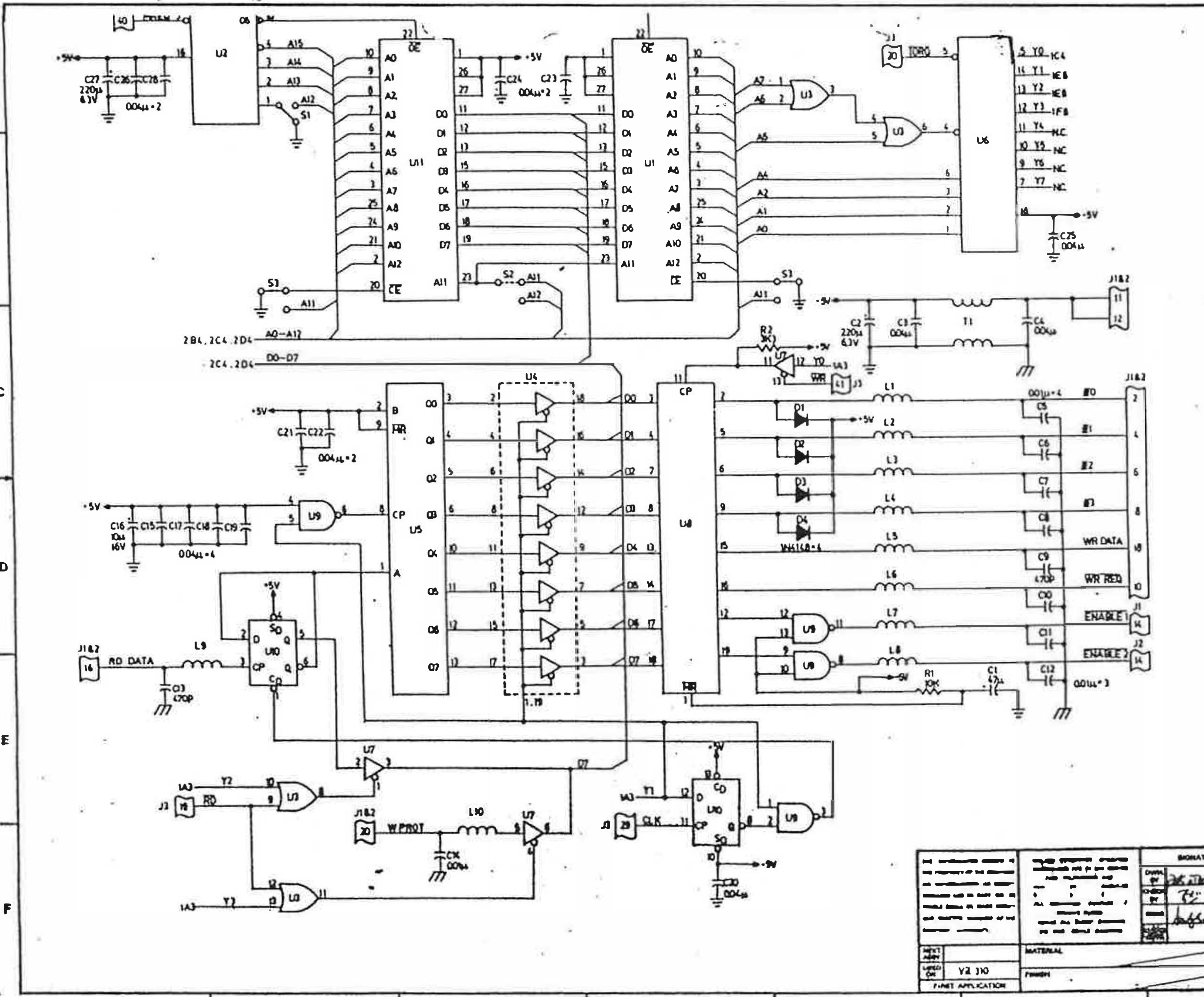
VIDEO TECHNOLOGY LTD

10TH OCTOBER, 1984.

Contents

Electrical Specification

Supply Voltage : +5V, 300mA (Supply from disk drive through flat cable)



5. U1 & U11 PIN OUT FOR 2764 ONLY.

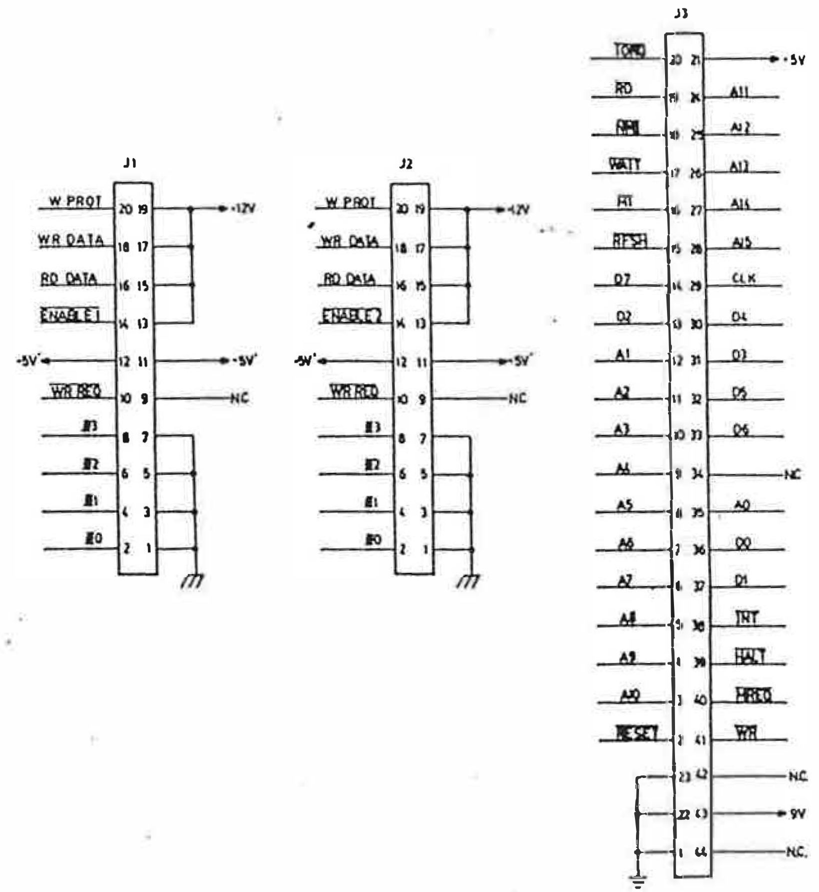
4. FOR 14 PIN I.C. PIN 7—GND, PIN 14—+5V.
16 PIN I.C. PIN 8—GND, PIN 16—+5V
20 PIN I.C. PIN 10—GND PIN 20—+5V.

3 FOR 2764, 2464, 2732, 2432, S3 CONNECTED TO GND
FOR 2364, S3 CONNECTED TO A11.

2 FOR 2764, 2464, 2732, 2432, S2 CONNECTED TO A11.
FOR 2364, S2 CONNECTED TO A12.

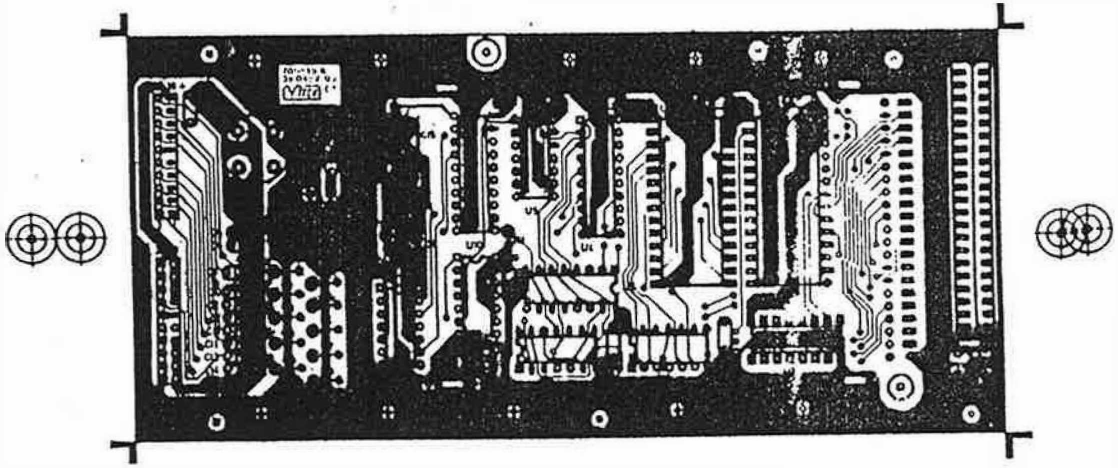
NOTE: 1 FOR 2364, 2764, S1 CONNECTED TO GND
FOR 2732, S1 CONNECTED TO A12

VIDEO TECHNOLOGY LTD		
TITLE	DISK CONTROLLER CIRCUIT DIAGRAM	
SIZE A1	CODE IDENT	DWG NO 65-0452-03
SCALE NONE		SHEET 1 OF 1
NEXT ASSY	USED ON V2 310	FIRST APPLICATION



SIGNATURE	DATE	VIDEO TECHNOLOGY LTD
DRAWN BY		TITLE DISK CONTROLLER CONNECTOR
CHECK BY		
SHOP		

VZ 310
FIRST APPLICATION
MATERIAL
FINISH
SIZE A1
DWG NO 86-0482-03



COMPONENT SIDE

VIDEO TECHNOLOGY LTD		
TITLE DISK CONTROLLER COMPONENT LAYOUT		
SIZE A1	CODE IDENT	DWG NO 65-0452-03
SCALE NONE		SHEET 3 OF 3

USED ON VZ 310

FIRST APPLICATION

MATERIAL

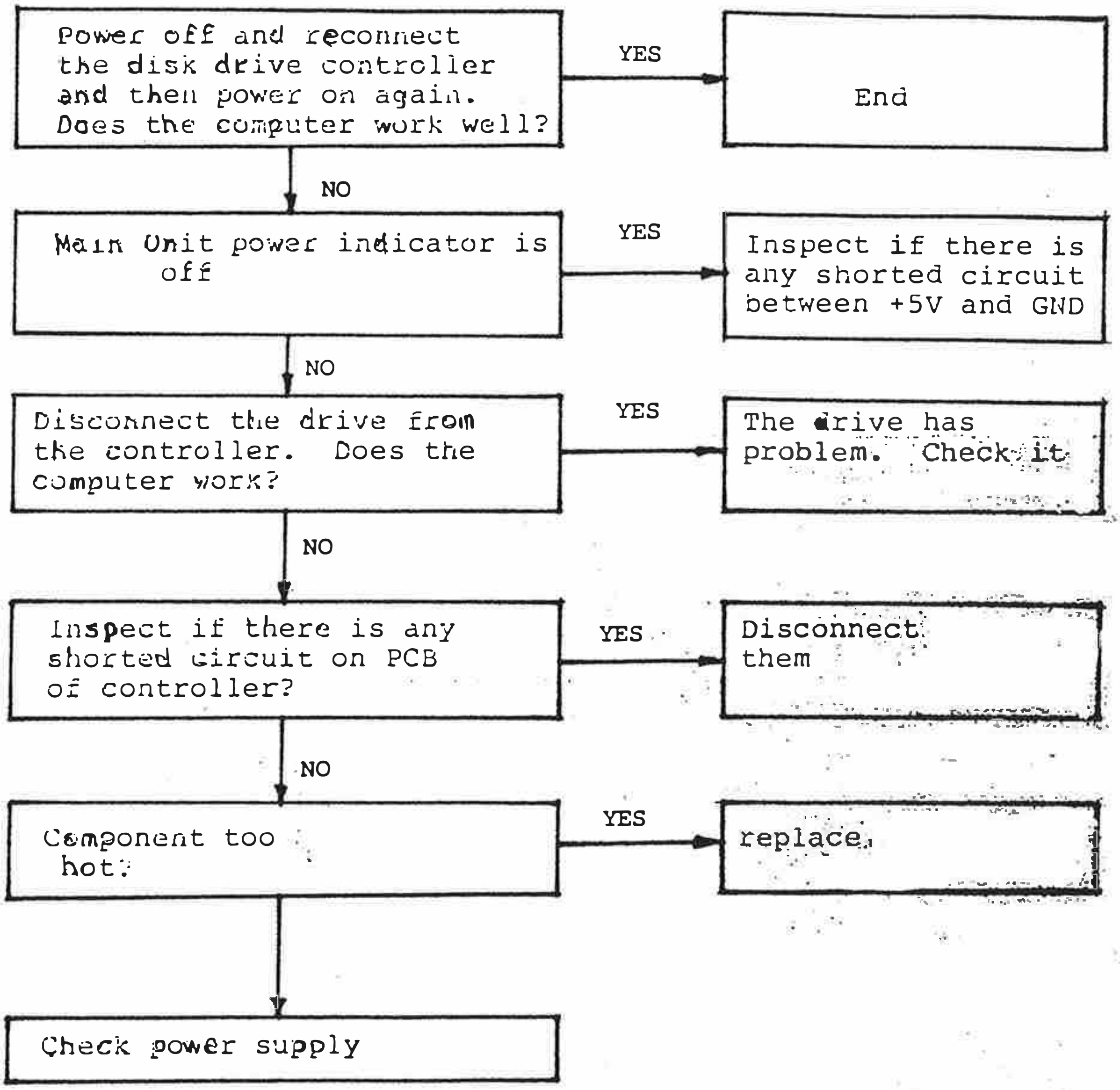
FINISH

4. Part List of DI-40 Disk Drive Controller

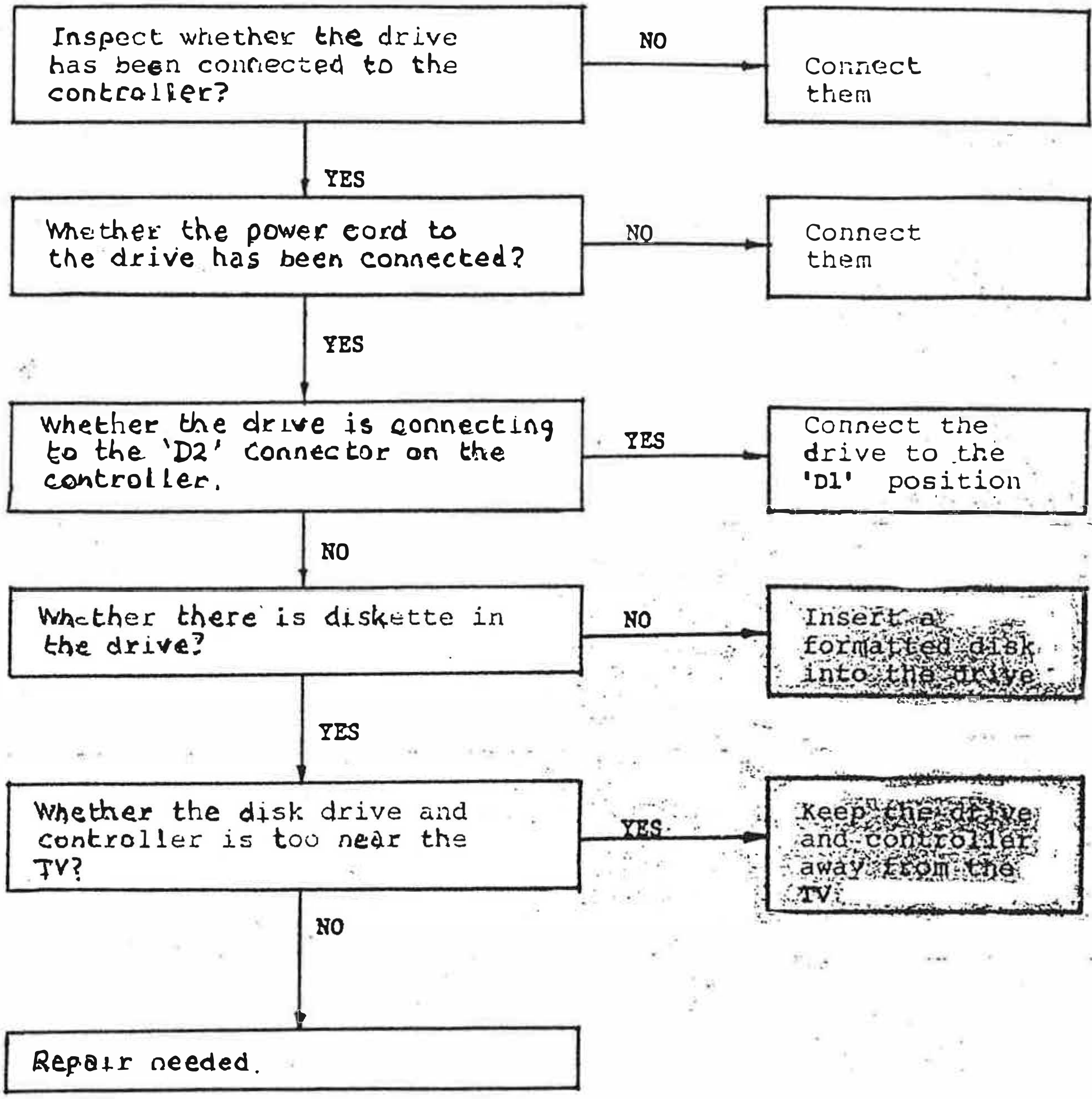
U1	2764
U2	74LS138
U3	74LS32
U4	74LS244
U5	74LS164
U6	74LS138
U7	74LS125
U8	74LS273
U9	74LS00
U10	74LS74

5. Trouble Shooting Guide

a) The computer cannot work after connecting the disk drive controller.



b) DOS Command does not work



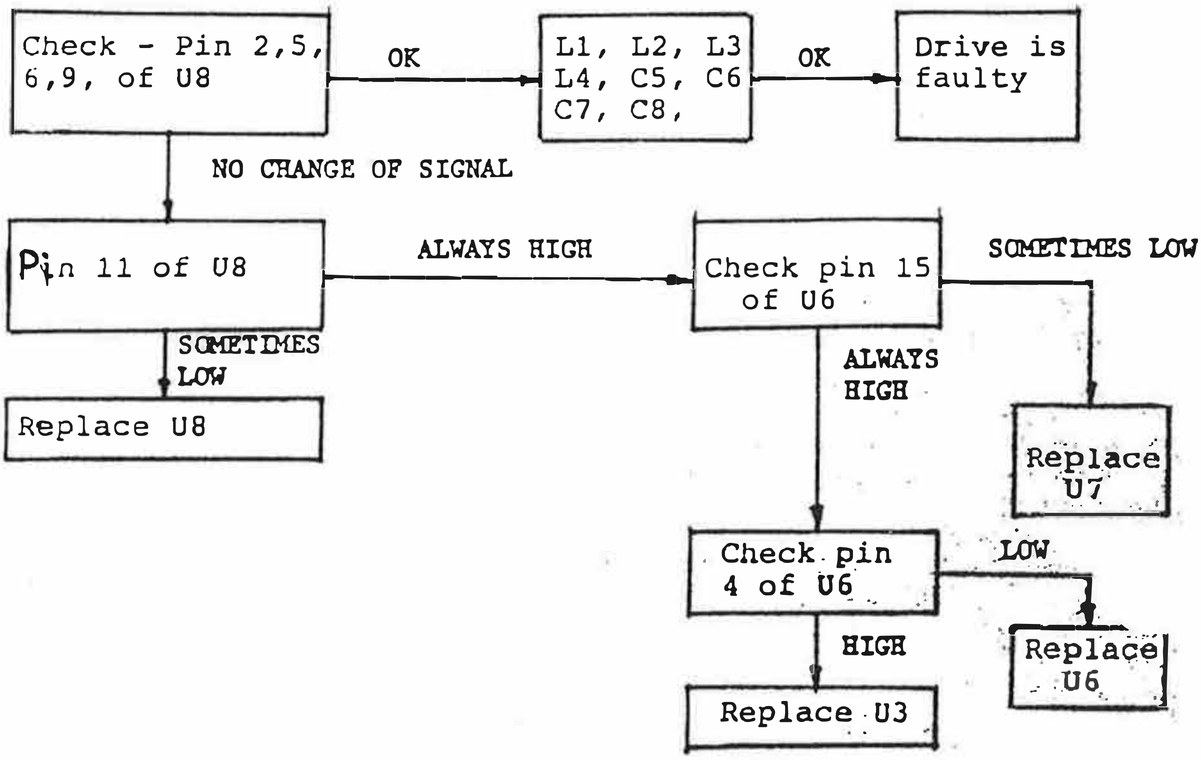
c) Repair guide

i/ R/W head does not move?

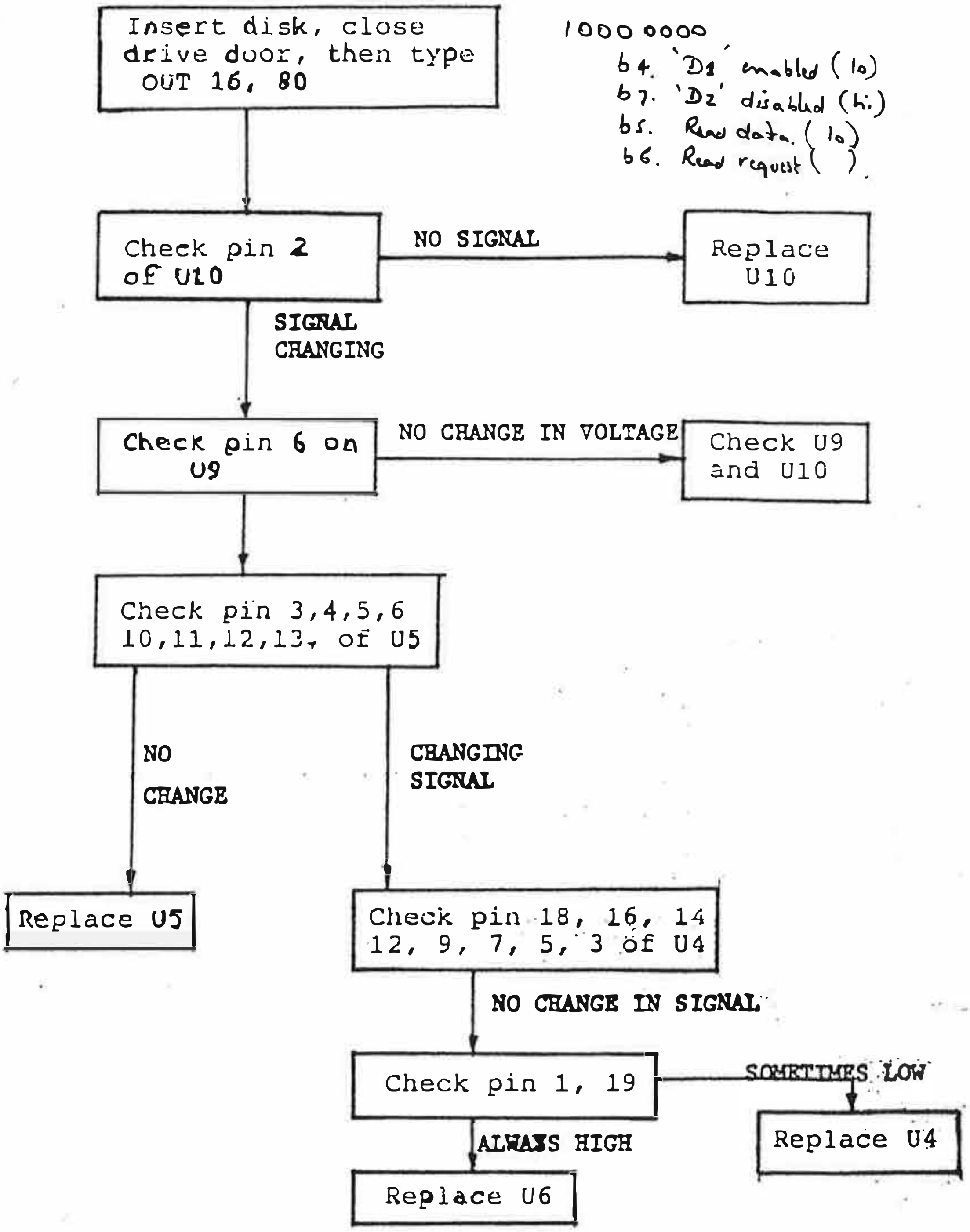
Type in the following program and run it

```
10 OUT 16, 81     1000 0001
20 OUT 16, 88     1000 1000
30 OUT 16, 84     1000 0100
40 OUT 16, 82     1000 0010
50 GOTO 10
```

change stepper motor control phases in bit 0-3. for 'D1'



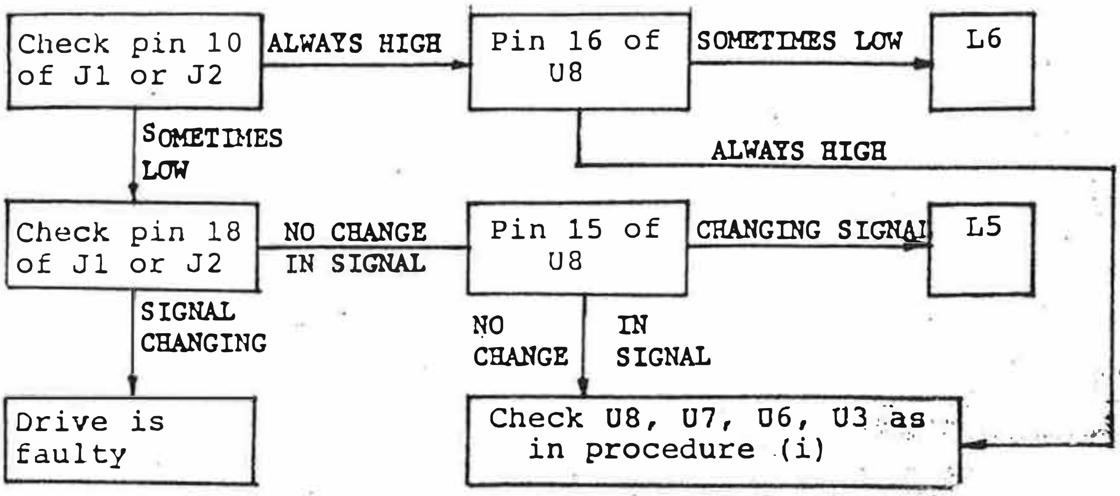
ii) Cannot read data from drive



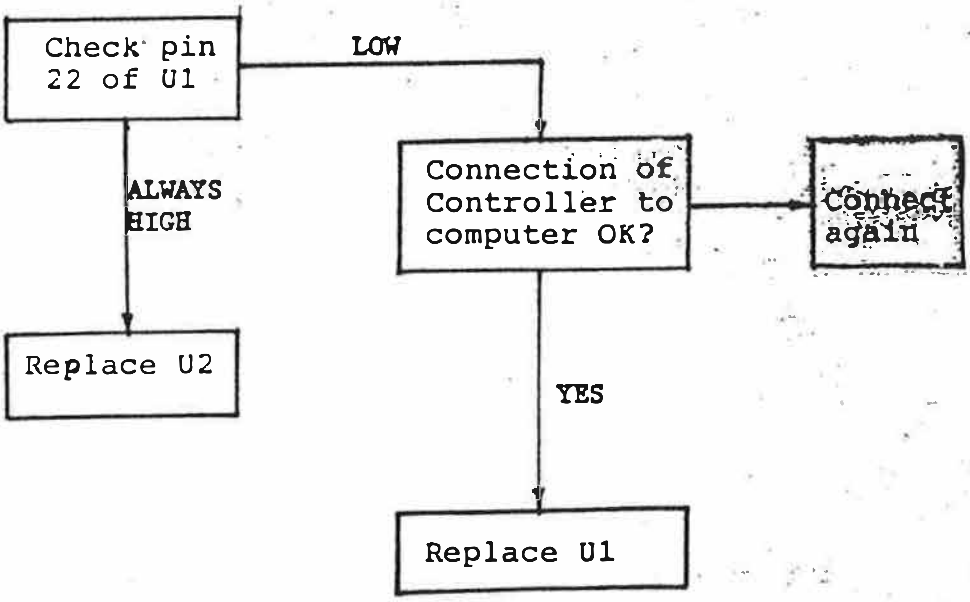
iii) Cannot write data to disk

Type in the following program

```
10 OPEN "TEST", 1
20 PR# "TEST", 1
30 GOTO 20
```



iv) DOS does not function




That's All
Folks!